Carte VME codeuse de charge : QDC16

O. Bourrion, J.C. Cerri, M. Koubaa

Carte VME codeuse de charge : QDC16

1 DESCRIPTION

La carte QDC16 a pour fonction d'effectuer des intégrations de charges sur 16 voies indépendantes et de les convertir en données 12 bits. Ces impulsions de courant sont négatives.

Elle est conçue au format VME avec un connecteur auxiliaire. Elle intègre aussi les fonctionnalités suivantes (tous les réglages sont individuels) :

- Réglages des piédestaux ;
- Suppression de voie avec seuils réglables;
- Sélection du calibre d'intégration ;
- Bufferisation de 25 événements ;
- Génération d'une interruption VME en mode coup par coup ;
- Échelle glissante ;
- Possibilité d'intégrer les voies en utilisant soit des portes individuelles, soit une porte commune ;

2 Spécifications

2.1 Générales

- Carte VME au format CERN ;
- Résolution 12 bits ;
- 16 voies indépendantes, individuellement sélectionnables ;
- Taille de chaque buffer de sortie : 512 mots (permet 25 évènements avec toutes les voies conservées) ;
- Possibilité de stocker plusieurs trames dans le buffer de sortie ;
- SNR (***S**ignal to **N**oise **R**atio*): <2 canaux a mi-hauteur ;
- Non linéarité intégrale : < 1% ;
- Non linéarité différentielle : à mesurer;
- Impulsion d'entrée négative (pas de positif) ;

2.2 Entrées-sorties

2.2.1 START (NIM)

Signal d'entrée. Signal de validation des portes et de déclenchement de conversion A/D.

2.2.2 RAZ (NIM)

Signal d'entrée. Cette entrée permet de stopper les intégrations en cours.

2.2.3 Busy (NIM)

Signal de sortie. Signale qu'aucun autre START ne doit être reçu par la carte QDC16 (risque de perte de données).

2.2.4 Gate Commune (NIM)

Signal d'entrée. Signal que l'on envoie au CCT (dépendant du chois de l'utilisateur),et qui détermine le temps d'intégration.

2.2.5 Gate Individuelles (NIM et ECL)

Signal d'entrée. Signal que l'on envoie au CCT (dépendant du chois de l'utilisateur),et qui détermine le temps d'intégration (2 standards disponibles).

2.2.6 Visualisation

2 LED sont implantées. Elles permettent la visualisation des états des signaux START, et de Busy, elles s'allument pour un minimum de 130 ms.

2.3 Caractéristiques des entrées analogiques

Les CCT ont :

Une dynamique d'entrée de 100µA à 100mA.
Une largeur minimum de gate de 10ns.
Signaux gate et reset en ECL.
Retard minimum entre le signal gate et l'impulsion d'entrée de 5ns.
Une dynamique de sortie de 0 à 3V.

2.4 Consommation

+5V : 1A

-5V : 1A

-12 V : inférieur à 200mA

2.5 Relation entre les signaux

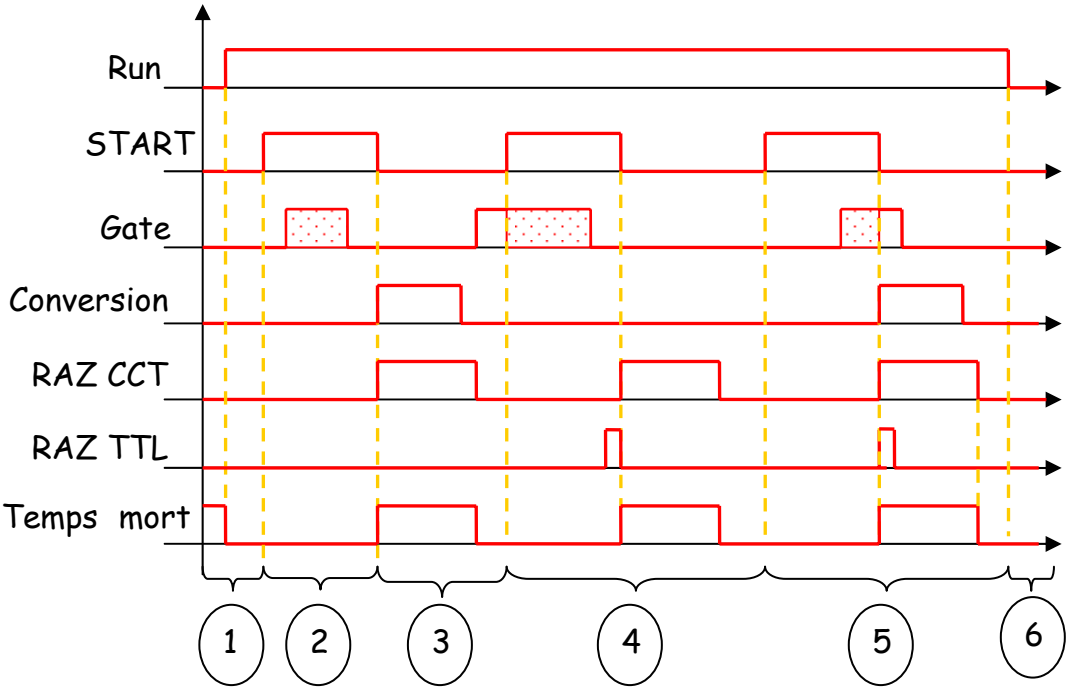


Figure 1 : Chronogramme temporel de fonctionnement

1 : La carte est activée par une écriture VME, alors RUN passe à l'état actif, et la carte sort du temps mort. Si elle est désactivée, la carte n'effectue aucune conversion.

2: Au moment où START passe à l'état 1, l'intégration est autorisée (Gates).

3: START passe à 0, l'arrêt des intégrations est forcé, et la conversion est déclenchée. La durée du temps mort dépend de la durée de conversion (→voir caractéristiques de l'ADC), et de la durée du raz CCT (programmable sur 8 bits).

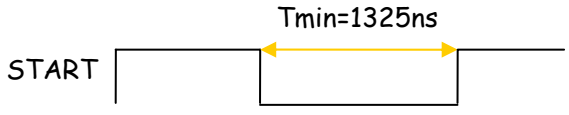
4 : Raz TTL reçu pendant START actif ➔ annule les intégrations en cours et inhibe la conversion.

5 : Raz TTL reçu en dehors du START actif ➔ signal ignoré.

6 : run passe à zéro ce qui signifie que la carte n'est plus en fonctionnement.

2.6 Relations Temporelles

Temps de START au niveau 0 logique minimum : 1325 ns.
Ce temps comprend le temps de conversion et d'évacuation des données présentes dans les buffers d'acquisitions (garanti par la simulation).



Délai mesuré entre le front descendant du START NIM et du front montant de BUSY : t_{dstm}=18.5ns

Délai mesuré entre l'arrivée du signal de :
Porte commune en ECL diff sur les CCT et la sortie : 10ns
Portes individuelles en ECL diff sur les CCT et la sortie : 5ns.

2.7 Façade avant

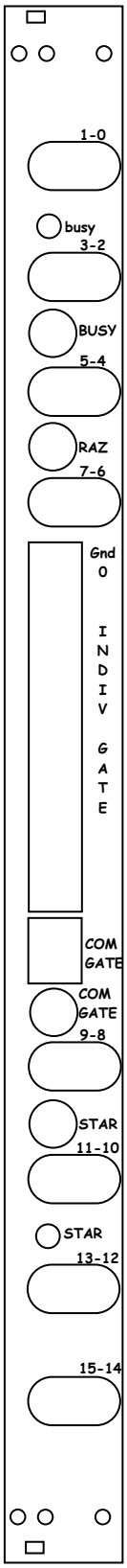


Figure 2 : Face avant de la carte

2.8 Placement des composants

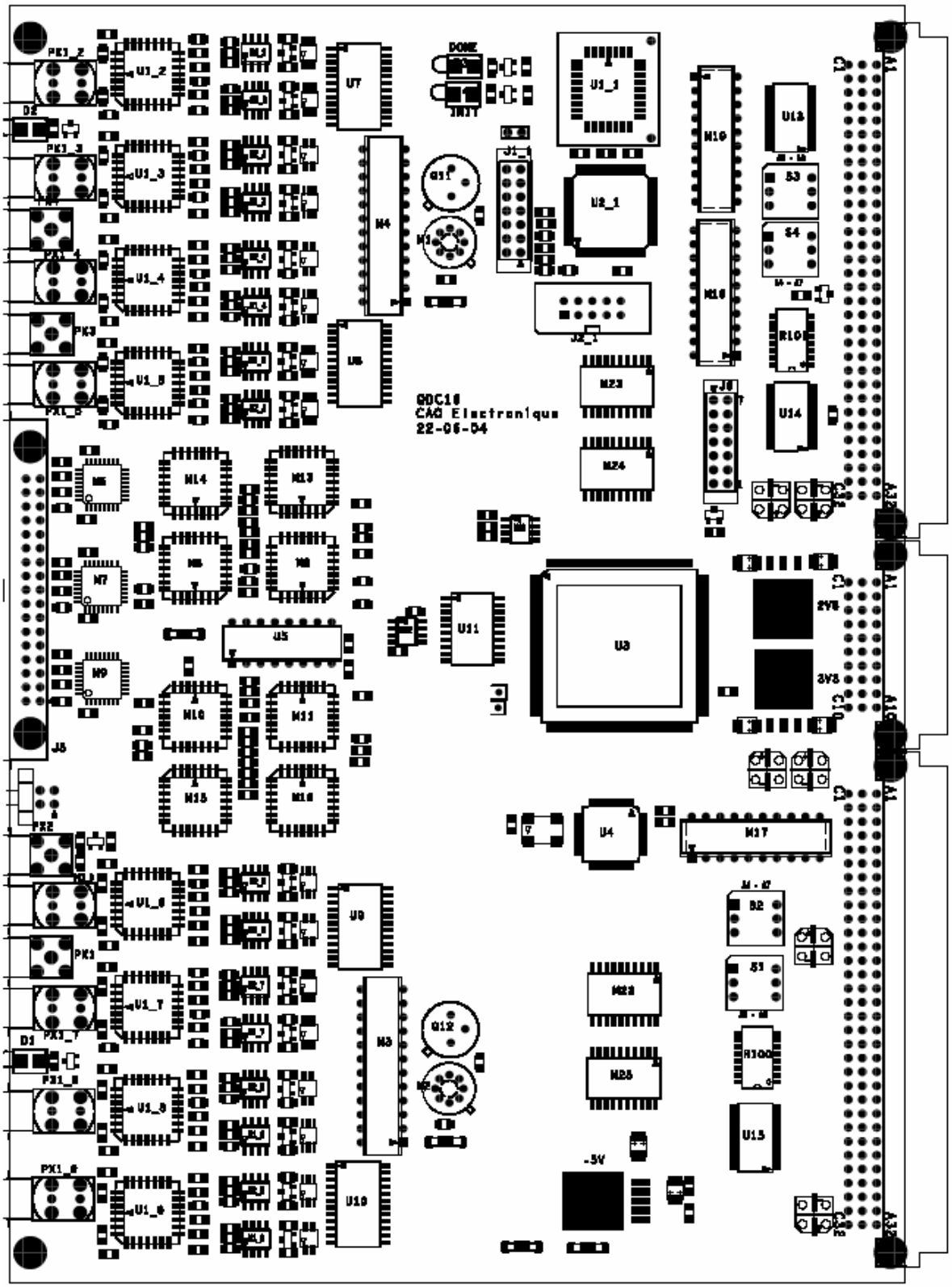


Figure 3 : Vue de dessus de la carte

2.9 Explication Fonctionnelle

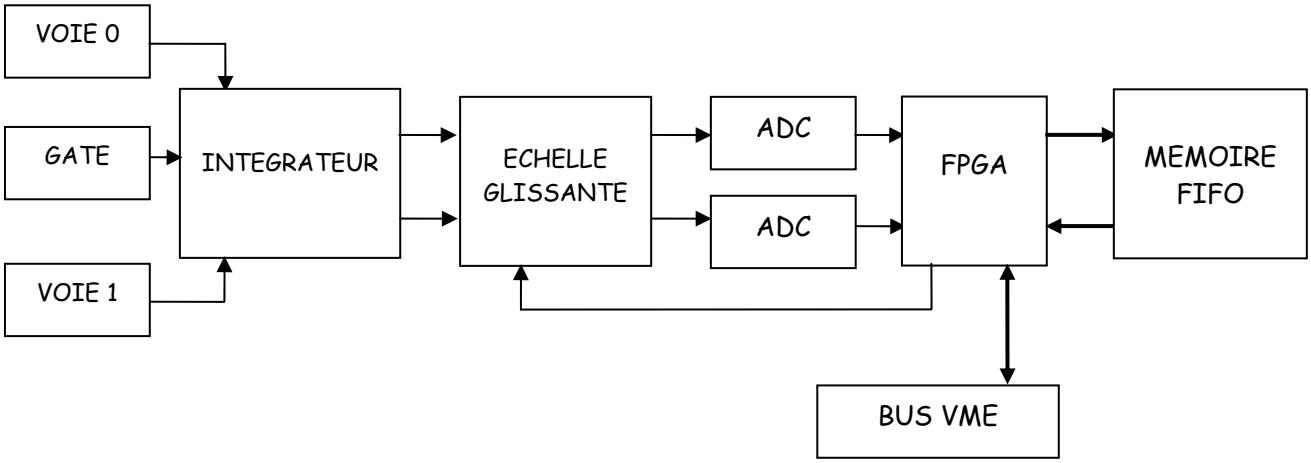


Figure 4 : Schéma général de fonctionnement

Un courant en entrée de chacune des voies est convertit à l'aide d'un intégrateur si un signal de porte est détecté. La charge étant convertie en une tension analogique, celle-ci va être numérisée grâce à un Convertisseur Analogique Numérique, ou ADC.
Un FPGA (Field Programmable Gate Array) récupère cette donnée numérique pour ensuite la traiter au moyen du programme synthétisé (en VHDL). Les valeurs gardées seront stockées dans une mémoire FIFO externe.
L'échelle glissante est présente pour corriger les erreurs de non linéarité différentielle des ADC, celle-ci sera plus détaillée par la suite.
Ce schéma reste néanmoins général. En effet, les grandes parties qui le composent seront développées dans ce rapport.

3 Fonctionnement

3.1 Entrées - Sorties

Trois standards de signaux peuvent être rencontrés : TTL, NIM et ECL.

1.1.1 16 voies analogiques

Signaux d'entrées. Les seize signaux sont intégrés pendant un temps qui dépendra de la durée des signaux de portes suivants afin d'obtenir la mesure.

3.1.1 Choix du type de porte et validation

Deux types d'entrées portes sont disponibles sur la carte : la porte commune et les portes individuelles. Les portes individuelles ont l'avantage de posséder chacune une largeur et un déclenchement temporel propre. Quant à la porte commune, elle permet d'avoir une intégration de même durée sur toutes les voies.
L'utilisateur a le choix entre ces deux types de porte, ce choix est exclusif. Ce choix étant fait, il faut le valider avec le START pour permettre les intégrations.
De plus, nous avons deux types de portes communes : NIM et ECL diff. ; il faudra donc prévoir une logique permettant le choix de celle-ci.

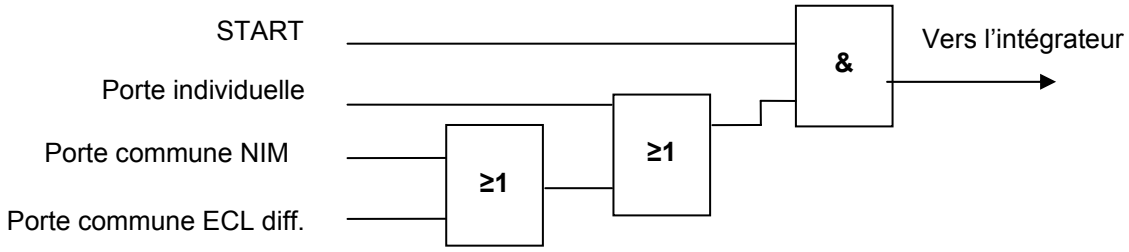


Figure 5 : Cellule d'orientation pour les choix des portes

La première porte OU permet le choix de travailler avec une porte commune en NIM ou en ECL diff. La seconde permet le choix entre la porte commune ou individuelle, et la dernière porte qui est un ET valide notre choix avec le START.

3.2 Compteur de START

C'est un compteur 32 bits incrémenté sur chaque front montant de START.

3.3 Compteur de voies touchées

A chaque fois que l'on compare une valeur à un seuil et que celle-ci est supérieure, le compteur est incrémenté. Sachant qu'il y a 16 voies, on peut rencontrer les cas allant d'aucune voie touchée à toutes d'où la mise en œuvre d'un compteur 5 bits.

3.4 Eléments de configuration de la carte

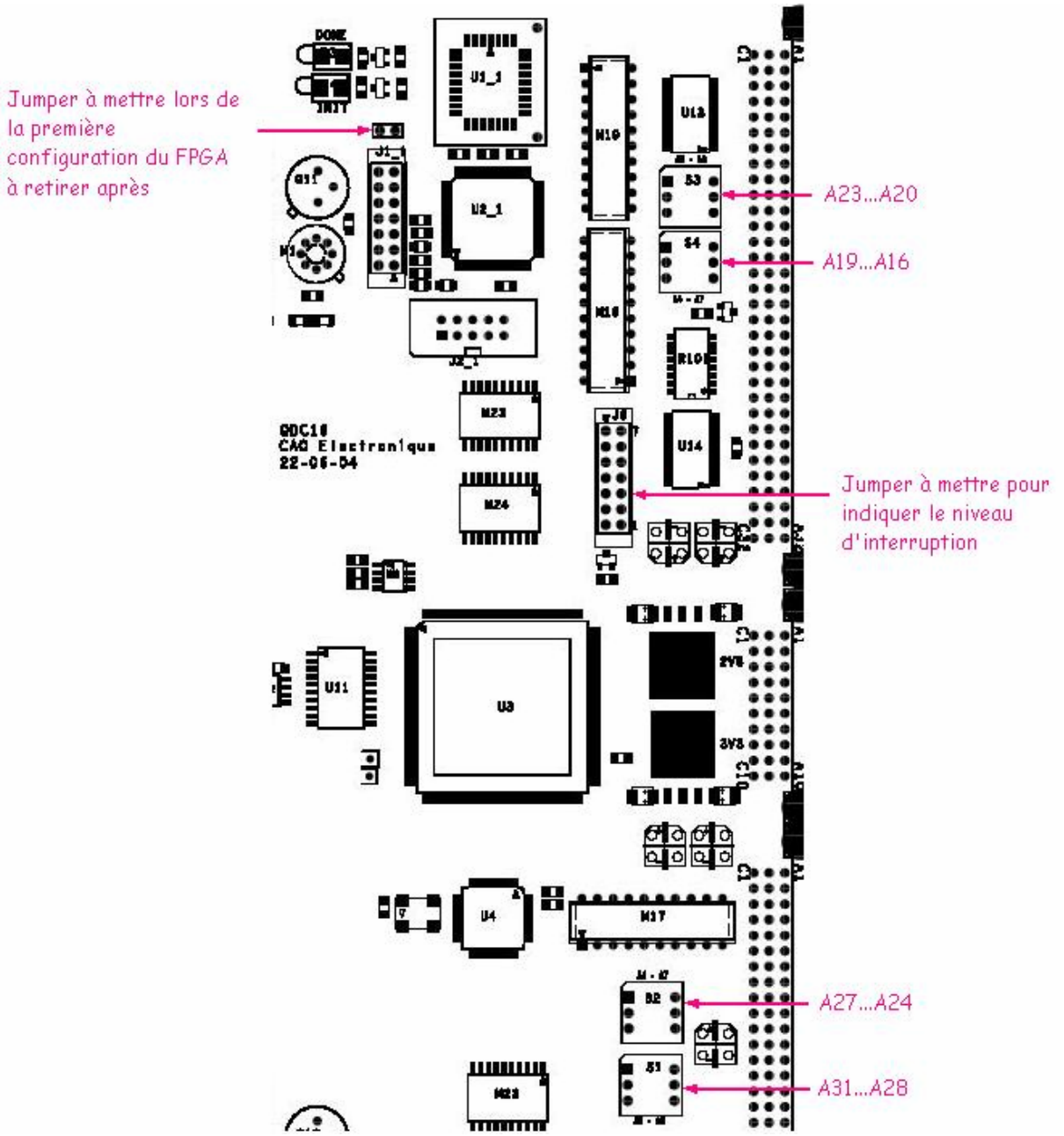


Figure 6 : Eléments de configuration de la carte

4 Échelle glissante

Les ADC ayant tous des erreurs de linéarité différentielle (Differential Non Linearity) non négligeables, une correction par échelle glissante est prévue.

4.1 Fonctionnement

La DNL de l'ADC est donnée par la documentation constructeur du composant. Elle est exprimée en nombre de LSB (LSB = Least Significant Bit). Un LSB donne une information très importante car il nous révèle la résolution de notre convertisseur.

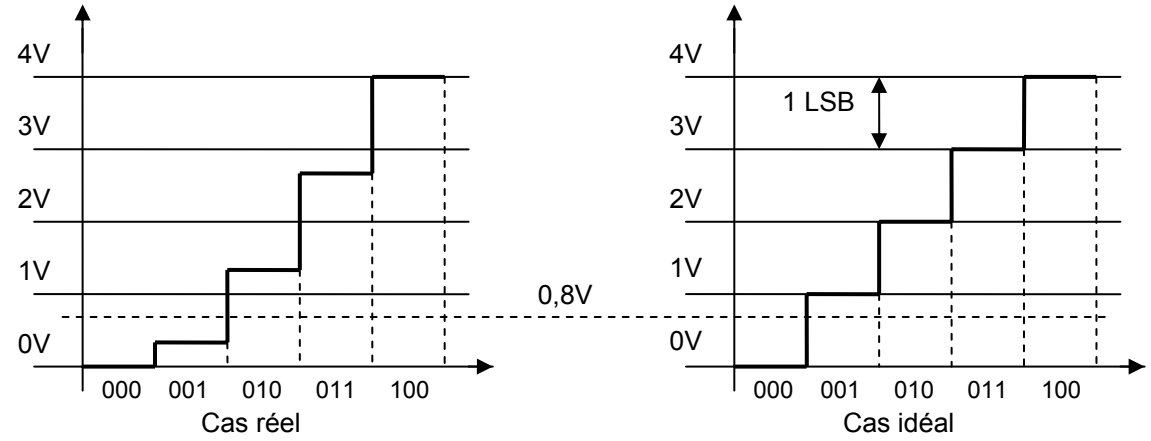


Figure 7 : Linéarité différentielle idéale et réelle

Dans le cas idéal, nous voyons que le LSB vaut 1V. Ce qui ne s'observe pas dans le cas réel. Voici un exemple pour comprendre :
Supposons que l'on ait 0,8V en entrée à convertir. Le cas idéal voudrait que l'on ait comme code en sortie 000, or en réalité nous aurons 001, ce qui naturellement représente une erreur si notre LSB vaut 1V ! Les valeurs par palier sont 1, 2, 3, N, donc en toute logique 0,8V aurait le même code que 0 car il est compris entre 0 et 1V.
Pour corriger ce problème, nous allons ajouter en entrée une valeur analogique égale à la valeur du LSB de l'ADC. Ensuite, après la conversion, il suffira de retrancher le code binaire correspondant à la valeur analogique correspondante.

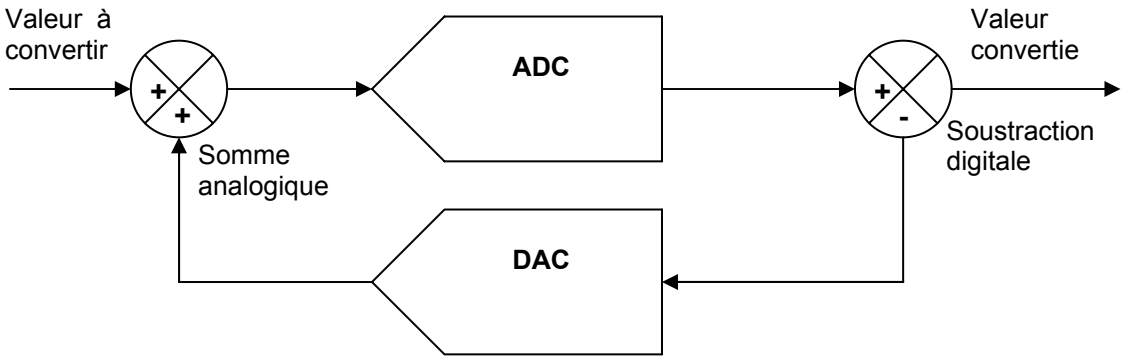


Figure 8 : Chaîne simplifiée de l'échelle glissante

Principe de fonctionnement : Ajoutons au 0,8V a convertir une valeur égale au LSB c'est-à-dire 1V en sortie du DAC. Notre convertisseur voit donc à son entrée une tension totale de 1,8V. Cette valeur est comprise entre 1V et 2V, donc le code en sortie de l'ADC sera 001. Ensuite on retranche le code binaire correspondant à la valeur analogique précédemment ajoutée (1V➔001). On revient donc à

faire la soustraction suivante : 001-001=000. Le résultat de cette soustraction est important car celui-ci représente le code de toutes les tensions comprises entre 0 et 1V.

Nous constatons bien qu'une correction a été effectuée. Mais ceci va dépendre aussi des irrégularités des DNL du convertisseur car nous aurons sans doute quelques résultats faux. On dira qu'en moyenne sur un nombre suffisant d'acquisitions nous obtiendrons un résultat correct.
Nous allons développer par la suite l'implantation de cette échelle glissante.

4.2 Réalisation

Afin de réaliser cette échelle glissante, plusieurs composants (analogique et numérique) sont nécessaires. L'ADC que nous avons est un 12 bits. Le DAC à choisir doit nécessairement être de résolution inférieure à 12 bits pour que le LSB du DAC soit supérieur au LSB de l'ADC. Si le LSB du DAC est supérieur au LSB de l'ADC, il devra être ajusté et pour cela nous avons implantons un système de gain variable (par potentiomètre numérique).
L'additionneur peut être simplement réalisé avec un montage AOP additionneur. Pour la soustraction (qui est numérique) nous décidons de la réaliser par soft à l'intérieur du FPGA.

La figure ci-dessous illustre la chaîne implantée.

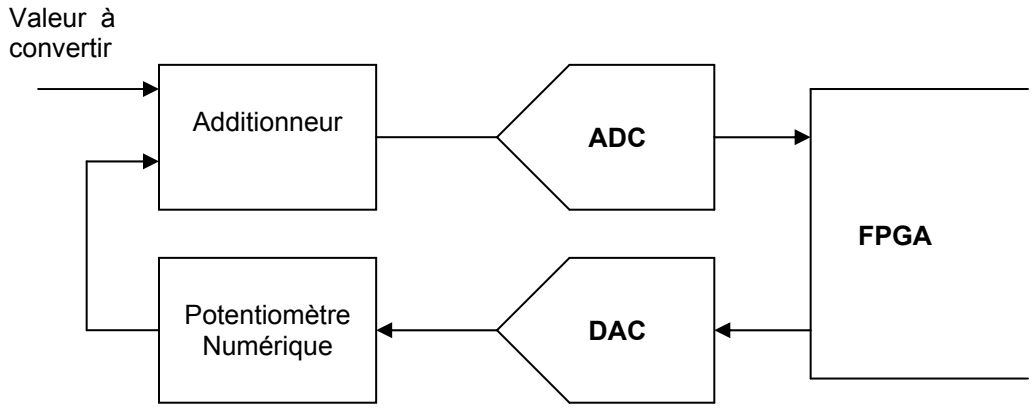


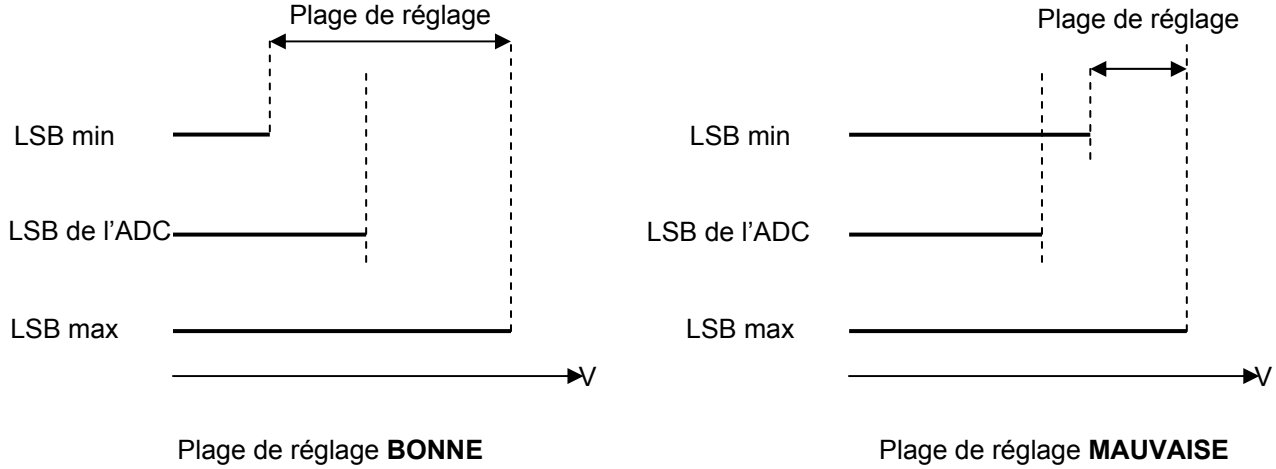
Figure 9 : Chaîne détaillée de l'échelle glissante

4.3 Calculs

Nous choisissons un DAC de résolution 8 bits. La résolution est :

$$LSB=\frac{\Delta Vmax}{2^N}$$

Comme dit précédemment, le gain variable est obtenu grâce à l'utilisation d'un potentiomètre numérique. Il faut dimensionner ce potentiomètre.
La figure ci-dessous illustre assez bien la variation de la plage de réglage du potentiomètre. Il faut que ce potentiomètre numérique soit dimensionné de telle sorte qu'il puisse couvrir une plage de réglage dans laquelle la valeur du LSB de l'ADC soit présente. Si le dimensionnement est trop faible, alors on risque d'être dans le cas de la seconde illustration et ainsi de ne pas pouvoir descendre plus bas que le LSB min.



Plage de réglage **BONNE** Plage de réglage **MAUVAISE**

Figure 10 : Visualisation des plages de réglage du potentiomètre

En conclusion, il faut :

$$LSB\ min < LSB\ ADC < LSB\ max$$

Nous allons donc devoir calculer les différents éléments de la chaîne. Calculons tout d'abord le LSB de notre ADC :

Notre Vmax correspond à 3.3V. Notre ADC est un 12 bits, donc on peut facilement calculer le LSB grâce à la relation vu précédemment :

$$LSB\,ADC = \frac{Vmax}{2^N} = \frac{3.3}{2^{12}} = 0.80566mV$$

Calculons le LSB du DAC, sachant qu'il est alimenté en 3.3V également et de résolution 8 bits.

$$LSB\,DAC = \frac{Vmax}{2^N} = \frac{3.3}{2^8} = 12.89mV$$

Nous devons donc dimensionner les résistances de l'additionneur afin que le LSB du DAC soit égal au LSB de l'ADC.

4.3.1 Additionneur

Nous avons opté pour un montage AOP simple. Celui-ci sera composé d'un montage additionneur inverseur suivi d'un montage inverseur.

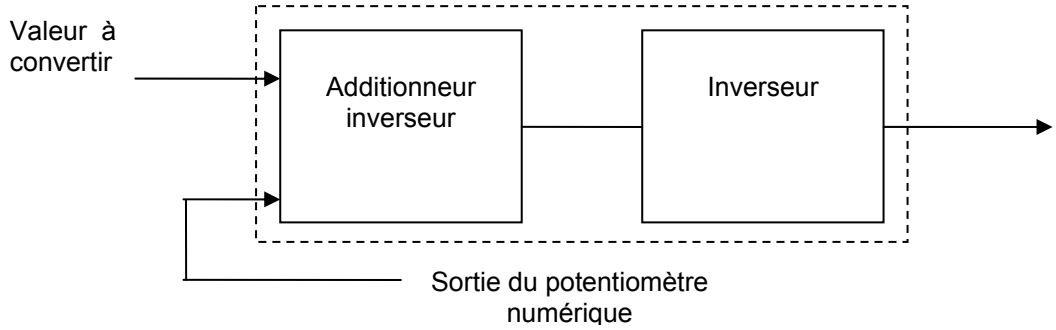


Figure 11 : Schéma bloc de l'additionneur

5 Schéma de câblage

Nous choisissons l'AD8058 qui a l'avantage de posséder deux circuits AOP intégrés.

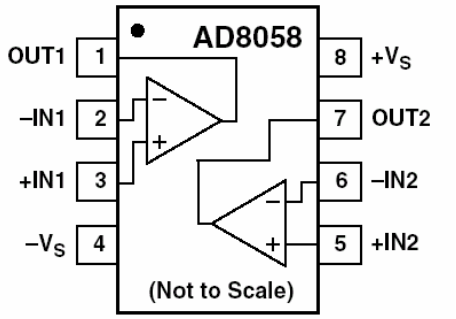
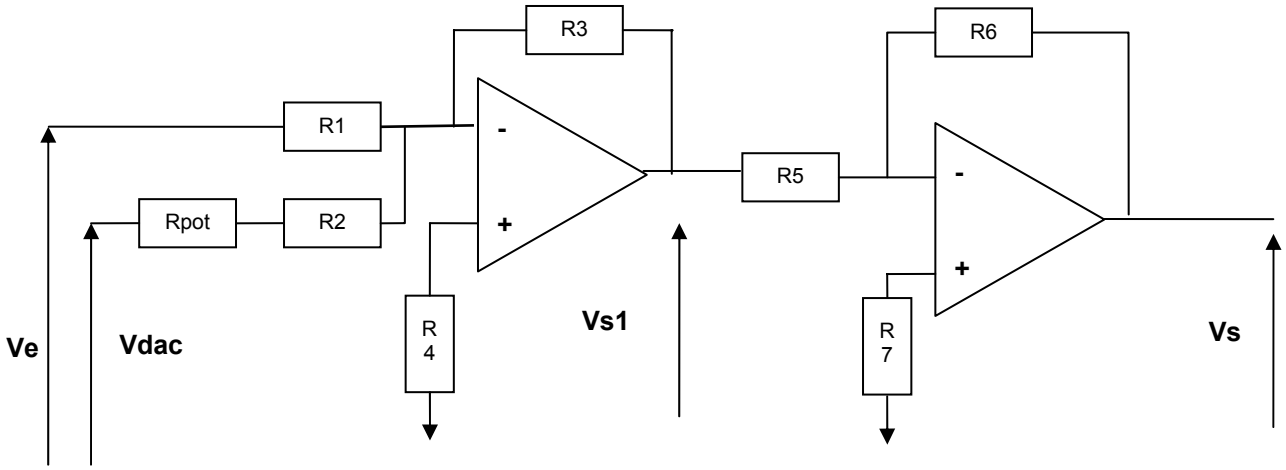


Figure 12 : Boîtier de l'AD8058

Voici le montage de l'ensemble additionneur et inverseur



Ve : Tension que l'on veut convertir
Vdac : Tension en sortie du DAC, tension qui va s'ajouter avec Ve
Vs : Tension de sortie

Figure 13 : Montage complet de l'additionneur

5.1 Equations

Voici donc les calculs pour trouver la fonction de transfert :

$$\frac{\mathrm{Vdac}}{\mathrm{R2}+\mathrm{Rpot}}+\frac{\mathrm{Ve}}{\mathrm{R1}}=-\frac{\mathrm{Vs1}}{\mathrm{R3}}$$

$$\mathrm{Vs1}=-(\frac{\mathrm{R3}}{\mathrm{R2}+\mathrm{Rpot}}\mathrm{Vdac}+\frac{\mathrm{R3}}{\mathrm{R1}}\mathrm{Ve})$$

L'inverseur s'écrit :

$$\mathrm{Vs}=-\frac{\mathrm{R6}}{\mathrm{R5}}\mathrm{Vs1}$$

En choisissant R6=R5, et R3=R1 nous obtenons la relation suivante :

$$\mathrm{Vs}=\frac{\mathrm{R3}}{\mathrm{R2}+\mathrm{Rpot}}\mathrm{Vdac}+\mathrm{Ve}$$

La sortie de l'ADC évolue si la tension analogique évolue d'une unité de valeur du LSB.
Ce que nous voulons, c'est avoir les LSB du DAC et de l'ADC identique, donc cela revient finalement a avoir cette relation :

$$\mathrm{LSB}_{\mathrm{ADC}}=\frac{\mathrm{R3}}{\mathrm{R2}+\mathrm{Rpot}}\mathrm{LSB}_{\mathrm{DAC}}$$

Donc :

$$\frac{\mathrm{LSB}_{\mathrm{ADC}}}{\mathrm{LSB}_{\mathrm{DAC}}}=\frac{\mathrm{R3}}{\mathrm{R2}+\mathrm{Rpot}} \qquad \text{Avec } \mathrm{LSB}_{\mathrm{ADC}}=\frac{3.3}{2^{12}} \text{ et } \mathrm{LSB}_{\mathrm{DAC}}=\frac{3.3}{2^{8}}$$

$$\frac{2^{8}}{2^{12}}=\frac{\mathrm{R3}}{\mathrm{R2}+\mathrm{Rpot}}$$

$$\frac{\mathrm{R3}}{\mathrm{R2}+\mathrm{Rpot}}=\frac{1}{16}$$

Avec les valeurs de Rpot, R2 et de R3 nous devons obtenir un rapport de 1/16 afin d'obtenir l'égalité que l'on recherche.

5.2 Choix des résistances

Concernant le choix des résistances, la plupart sont faciles à déterminer. En effet fixons par exemple R1=R3=R5=R6=4,7kΩ, les résistances R4 et R7 étant connectées à la borne positive, elles doivent avoir la même impédance que voit l'entrée négative. Étant donné que ces entrées négatives voient 2X 4,7kΩ en parallèle, la résistance à choisir est de l'ordre de 4,7kΩ/2=2,35kΩ. la valeur implantée est 2,55kΩ

Remarque : nous n'avons pas tenu compte des résistances Rdac (impédance de sortie des DAC) négligeables devant R2.

5.2.1 Potentiomètre numérique

Nous rappelons que nous devons obtenir : $\frac{R3}{R2+Rpot} = \frac{1}{16}$

On a donc : $R2 + Rpot = 16 * R3$

$R2 + Rpot = 75.2k\Omega$

Pour faire un choix, il faut regarder ce que l'on peut avoir comme valeur disponible pour le potentiomètre. Nous avons trouvé un boîtier qui comprend 4 potentiomètres numériques, et dont le chargement des données se fait de manière série.

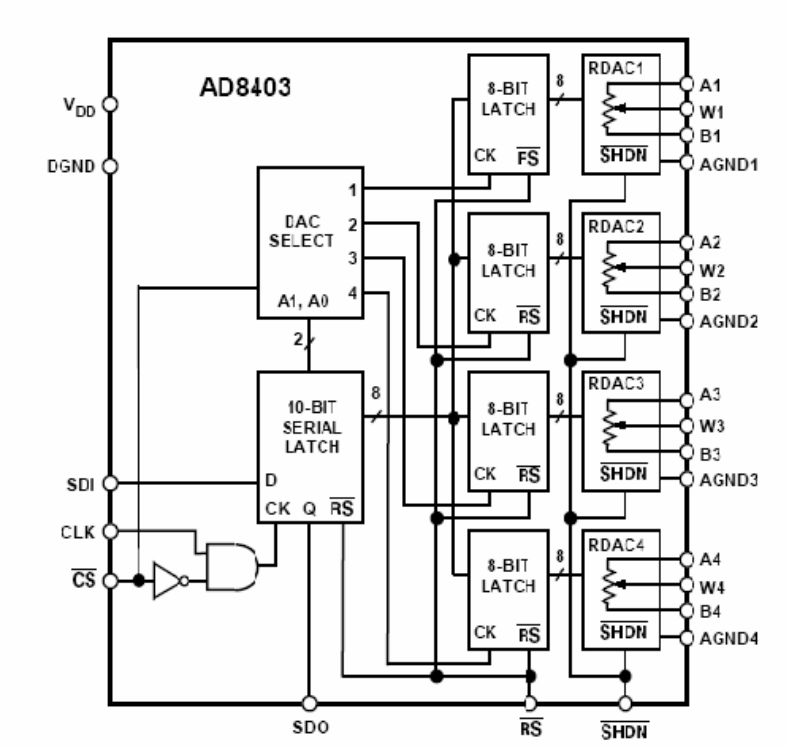


Figure 14 : Diagramme bloc du AD8403

Celui-ci est très avantageux car étant donné que nous devons corriger les DNL de 16 ADC (car 16 voies) l'encombrement serait minime car nous aurons en tout 4 boîtiers.
De plus, l'écriture dans les registres se fait de manière série en cascadant les 4 boîtiers.

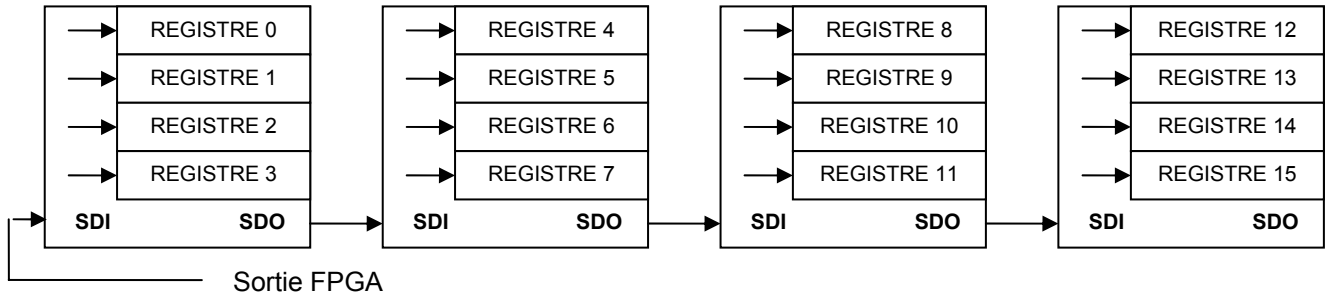


Figure 15 : Chaînage série des 4 circuits AD8403

Ce composant est disponible avec des valeurs de résistances de 1kΩ, 10kΩ, 50kΩ et 100kΩ. Naturellement la 100kΩ est à exclure de notre choix car elle est supérieure à 75,2kΩ.
Il reste donc trois choix de résistances possible, et pour ce faire nous allons étudier un paramètre qui est la dérive en température.

5.3 Dérive en température

5.3.1 Définition

La température est un facteur essentiel parfois pour le choix d'un composant. En effet elle peut avoir une influence non négligeable si l'on travaille dans un environnement assez affirmé en température. Certains composants sont plus sensibles que d'autres.

La dérive en température, ou coefficient de température est exprimée en ppm (partie par million). Dans la documentation technique du potentiomètre voici ce que l'on peut lire :

Resistance Tempco	$\Delta R_{AB}/\Delta T$	$V_{AB} = V_{DD}$, Wiper = No Connect	500	ppm/°C

Notre coefficient de température est de 500 ppm/°C. Cela veut dire que la valeur de la résistance augmente de 0,05% par degré.

Nous voulons que les effets de la température sur le potentiomètre soient négligeables car il faut toujours assurer un rapport de 1/16 même à une température élevée (+20° de la T° ambiante).Nous avons donc intérêt à choisir un cas où R2 est supérieure à Rpot afin de minimiser l'effet de la dérive de température.

De plus, il nous faut une résolution la plus importante possible sur la valeur de la résistance Rpot. Il faut donc choisir Rpot la plus faible possible.

Le meilleur compromis est l'AD8403 à 10kΩ.

Si Rpot=10kΩ :

R2=75,2-10/2=70,2kΩ

Ainsi avec R2 choisie à 70kΩ, la plage de variation de gain englobe bien 1/16.

5.4 Limitation de tension

Nous allons prévoir une limitation en tension en entrée de chaque ADC. En effet, la sortie de l'additionneur peut excéder une valeur limite que l'ADC ne supportera pas. La tension maximum supporté en entrée est donnée par le constructeur :

ABSOLUTE MAXIMUM RATINGS[1]

(T_A = 25°C, unless otherwise noted.)

V_{DD} to GND	−0.3 V to +7 V
Analog Input Voltage to GND	−0.3 V to V_{DD} + 0.3 V
Digital Input Voltage to GND	−0.3 V to +7 V
Digital Output Voltage to GND	−0.3 V to V_{DD} + 0.3 V

Table 1 : Tensions maximum recommandées du AD7476A

Nous alimentons notre ADC avec VDD=3.3V, donc la tension maximum que supportera une entrée sera de VDD+0.3V=3.6V
Nous allons simplement utiliser une diode Schottky pour limiter notre tension. L'avantage c'est que ce type de diode possède une tension de seuil de l'ordre des 0.3V.Nous allons en prévoir deux montées tête bêche afin que l'on limite aussi lorsque l'on est en négatif.

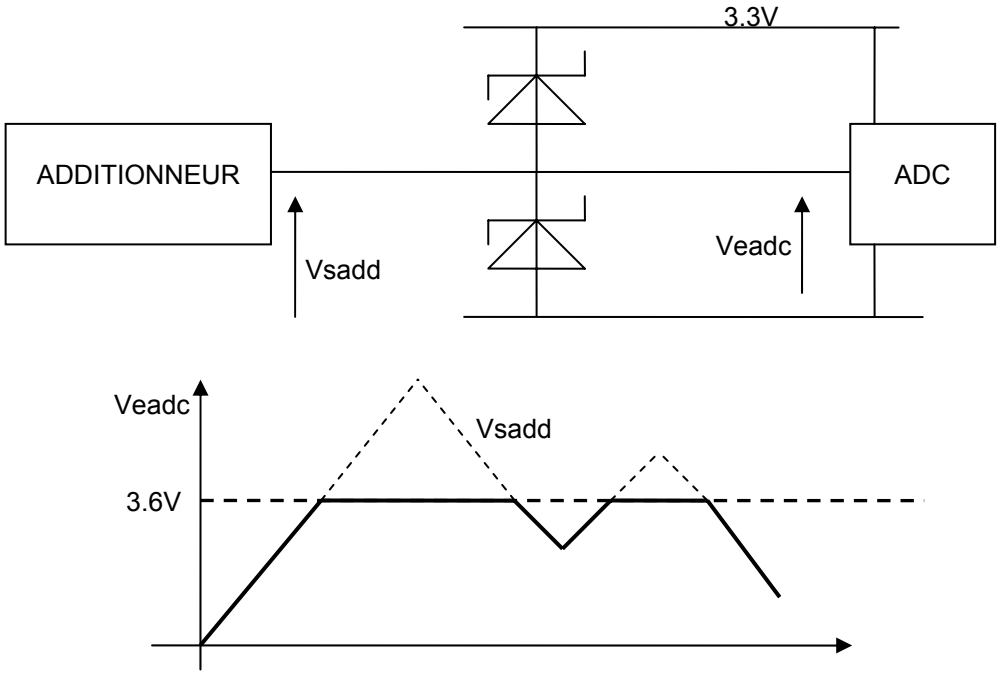


Figure 16 : Limitation de tension à diode Schottky

La diode commence à conduire dès que son anode est supérieure à celle de la cathode + 0.3V. La cathode étant reliée au 3.3V la diode conduira si Vsadd est supérieure ou égale à 3.6V. C'est limite donc il faut regarder la tension de seuil.

6 LE FPGA

6.1 Présentation du FPGA

Le FPGA utilisé est le Xilinx Spartan II et nous choisirons la version XC2S30 présentée ci-dessous dans un extrait de la documentation constructeur.

Table 1: **Spartan-II FPGA Family Members**

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O(1)	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	132	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	196	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

Celle-ci possède un nombre d'entrée sortie que nous estimons suffisant (132). De plus elle était disponible en magasin du laboratoire.

6.2 Alimentations du FPGA

Pour ce FPGA, il faut prévoir deux alimentations : l'alimentation du cœur du circuit en 2.5V et l'alimentation pour le standard en 3.3V vu précédemment.

Recommended Operating Conditions

Symbol	Description		Min	Max	Units
T_J	Junction temperature(1)	Commercial	0	85	°C
		Industrial	–40	100	°C
V_{CCINT}	Supply voltage relative to GND(2,5)	Commercial	2.5 – 5%	2.5 + 5%	V
		Industrial	2.5 – 5%	2.5 + 5%	V
V_{CCO}	Supply voltage relative to GND(3,5)	Commercial	1.4	3.6	V
		Industrial	1.4	3.6	V
T_{IN}	Input signal transition time(4)		-	250	ns

Table 2 : Alimentations recommandées du FPGA

Pour ce faire nous implantons des régulateurs afin d'obtenir nos deux tensions voulues.

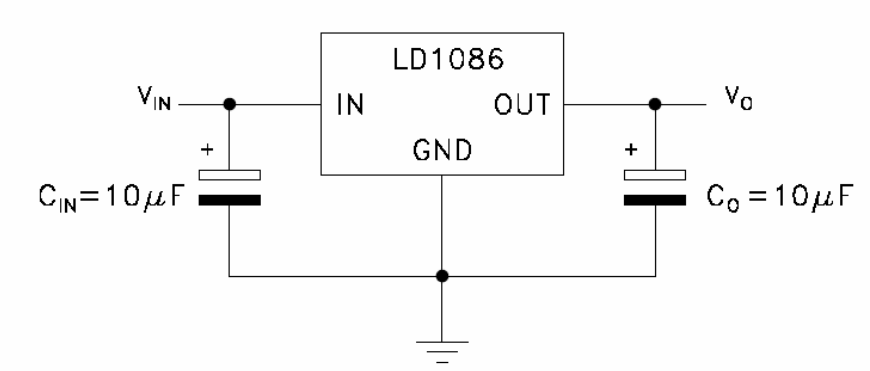


Figure 17 : Schéma du régulateur

Pour obtenir une alimentation de 2,5V, on a choisi le régulateur LD1086V25 et de le câbler comme ci-dessus. Même chose pour l'alimentation du 3,3V, seulement en utilisant le LD1086V33.

7 Les différents standards

7.1 Conversion NIM→ECL

Pour pouvoir convertir nos entrées NIM en ECL, on utilise le montage suivant :

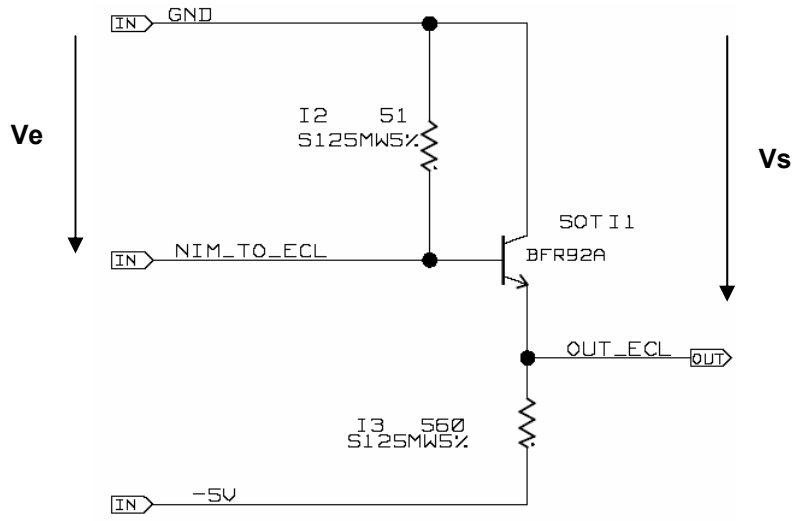


Figure 18 : Conversion NIM→ECL

Niveau Logique	Tension de sortie	Niveau Logique	Tension de sortie
0	0V	1	-0,8V
1	-0,8V	0	-1,7V
NIM		**ECL**	

Le premier point important est que ce montage est inverseur logique. On a déterminé la résistance de 560 Ohms telle de manière à obtenir un Vbe de 0,8V. Ce décalage de 0,8 Volts induit que lorsque la tension d'entrée vaut 0 Volt, on a en sortie -0,8V. Et quand il y a -0,8V, la sortie sera équivalente à -1,6V.

7.2 Conversion NIM→TTL

Pour pouvoir convertir l'entrée START NIM en TTL, on utilise le montage suivant :

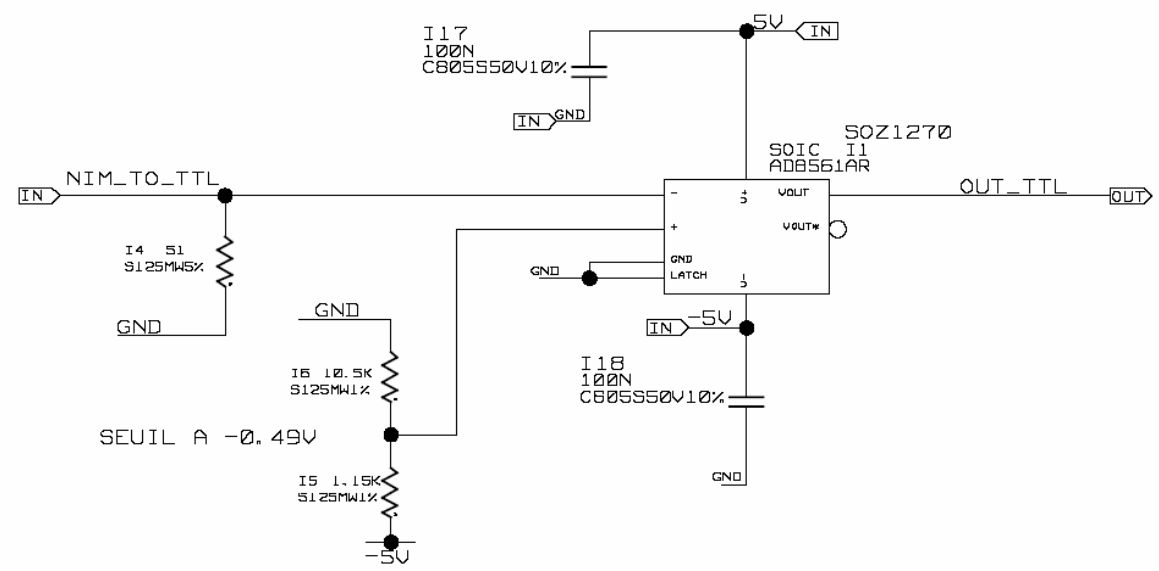


Figure 19 : Conversion NIM→TTL

7.2.1 Conversion TTL→NIM

Pour pouvoir convertir du TTL en NIM, on utilise le montage suivant :

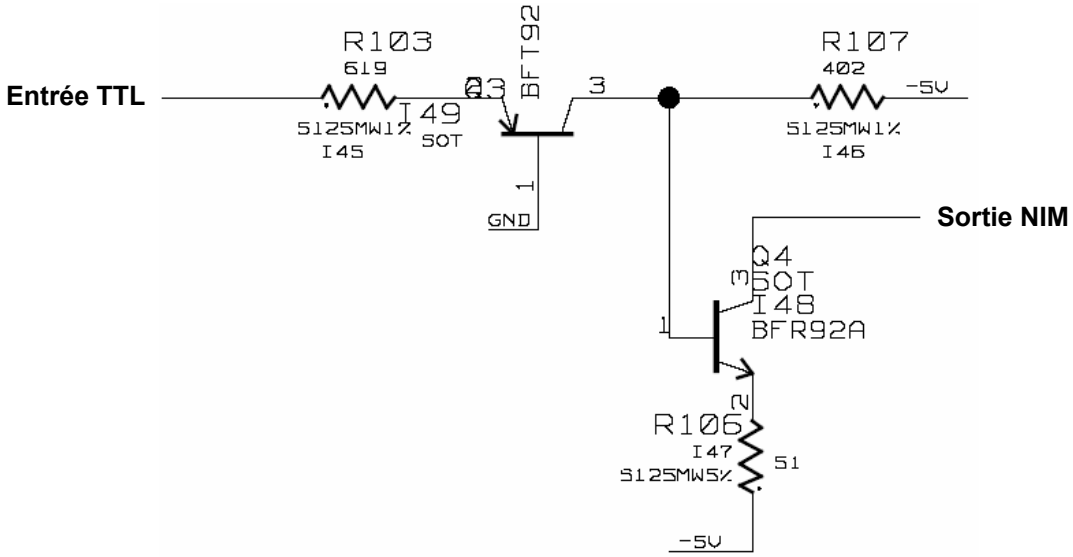


Figure 20 : Conversion TTL→NIM

Tableau Récapitulatif :

ECL		TTL		NIM	
0 Logique	*1 Logique*	*0 Logique*	*1 Logique*	*0 Logique*	*1 Logique*
-1,75V	*-0,8V*	*0V*	*5V*	*0V*	*-0,8V*

8 Interface utilisateur VME

8.1 Modificateur d'adresse reconnus

Les AM (Adresse Modifier) reconnus sont :
0F : accès aux données en mode étendu, privilégié (A32)
0D : accès aux données en mode, privilégié (A32)
0B : accès aux données en mode étendu, non privilégié (A32)
09 : accès aux données en mode étendu, non privilégié (A32)

Génération du CS interne :

```
CS_int<='1' when (BUS_A31_A24_OK='0' and BUS_A23_A16_OK='0' and
BUS_MODE_OK='0' and EPLD_VME_IACK='1' and EPLD_VME_DS0='0' and
EPLD_VME_AS='0') else '0';
```

8.2 IDCODE (read only 32)

Une lecture à cette adresse permet de savoir si on adresse à cette carte.

ID_CODE (32 bits)		
31	...	0

La valeur attendue à la lecture est $(16\ 79\ 14\ 83)_{hex}$.

8.3 FIFO_Readout : (read only 32)

Les données seront écrites dans la FIFO comme suit :

C'est pourquoi quand on lit, on récupère :

en premier le numéro de START sur 32 bits,

en deuxième le nombre de voies touchées sur 5 bits (tous les autres bits sont mis à 0),

ensuite on va envoyer les valeurs et les numéros de voie correspondants sous cette forme :

FIFO_Readout (32 bits)			
Numéro Voie (4 bits)	Valeur (12 bits)	Numéro Voie (4 bits)	Valeur (12 bits)

Il faut savoir que le nombre de lecture possible dépend du nombre de voies touchées. S'il est pair, on doit tout prendre compte en sachant qu'on récupère deux valeurs de deux voies différentes à chaque accès VME. Par contre, s'il est impair dans la dernière lecture de la fifo, on écrira deux fois la même chose (numéro de voie et valeur). Dans ce cas là, on doit prendre en compte qu'une partie envoyée (soit les LSB, soit les MSB ça n'a pas d'importance).

Exemple de 3 trames successives :

Ox00000001 *(numéro de START)*			
Ox00000003 *(nombre de voies touchées)*			
Ox2	Ox012	Ox0	Ox001
À jeter	À jeter	Ox4	Ox014
Ox00000002 *(numéro de START)*			
Ox00000004 *(nombre de voies touchées)*			
Ox3	Ox144	Ox1	Ox041
OxF	Ox714	Ox9	Ox600
Ox0000 0003 *(numéro de START)*			
Ox0000 0000 *(nombre de voies touchées➔ pas de données)*			

8.4 Module mise en route acquisition et adressage: (Write only 32)

Une écriture à cette adresse permet :

- de valider le run (carte en fonctionnement ou pas),
- d'effacer la FIFO,
- de préciser la durée d'effacement du CCT,
- de valider l'échelle glissante.
- de configurer la valeur envoyée vers le dac lorsque l'échelle glissante est désactivée.

Reg_sl_sc_and_run (32 bits)											
Undefined			DAC_offset (8 bits)			Val_sliding_scale (1 bit)	Duree_raz_CCT (8 bits)			Raz Fifo (1 bit)	Run (1 bit)
31	…	19	18	…	11	10	9	…	2	1	0

Si le **run** est à 1 la carte est en fonctionnement. Après que le run soit passé à l'état actif, on a une chance sur deux pour que l'ADC soit éteint, de plus, l'échelle glissante n'est pas non plus prête lors de la première mesure.
Conclusion : il faut jeter la première mesure après un RUN à cause de ces deux phénomènes.

Si le **Raz Fifo** est à 1, on reset la FIFO (ce bit n'est pas monostable, il faut réécrire 0 pour sortir la FIFO du reset)
On configure **Duree_raz_CCT** sur 8 bits par pas de 25ns, soit au max 255*25ns=6.375µs.
Val_sliding_scale :L'échelle glissante est validée lorsque le bit 10 est à 1.
DAC_offset : La valeur qu'on envoie vers le DAC quand l'échelle glissante est désactivée est comprise entre 0 et 255. (Cette valeur n'est pas retranchée par la suite car Val_sliding_scale =0).
Les bits restant sont non utilisés.

8.5 Potar_num : (write only 32)

On dispose de 16 potentiomètres numériques et on utilise une seule adresse codée sur 32 bits. On dispose de quatre boîtiers contenant chacun quatre potentiomètres numériques. Pour en configurer un seul, on a besoin d'envoyer une valeur codée sur 8 bits et une adresse (correspondant à sa place dans le boîtier) sur 2 bits.

Potar_num (1ere écriture)																					
Indice (1 bit)	Undefined (11 bits)						Adresse voie (2 bits)			Valeur (8 bits)				Adresse (2 bits)			Valeur (8 bits)				
31	30	…	20	19	…	18	17	…	10	9	…	8	7	…	0						
0	XXXX							Pour potar1						Pour potar0							

Potar_num (2ème écriture)																					
Indice (1 bit)	Undefined (11 bits)						Adresse voie (2 bits)			Valeur (8 bits)				Adresse (2 bits)			Valeur (8 bits)				
31	30	…	20	19	…	18	17	…	10	9	…	8	7	…	0						
1	XXXX							Pour potar3						Pour potar2							

Ce qui doit être fait : changer le bit indice à chaque accès VME et configurer d'abord les potentiomètres à l'adresse 00 de tous les boîtiers puis 01, 10 et 11. Une écriture VME dans le registre avec le bit indice à 1 déclenche effectivement l'écriture dans les potentiomètres.

Donc pour écrire les 16 voies, il faut faire 8 écritures 32 bits, comme suit

Potar_num (1ere écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
0	XXXX	00	Voie4	00	Voie0
Potar_num (2ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
1	XXXX	00	Voie12	00	Voie8
Potar_num (3ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
0	XXXX	01	Voie5	01	Voie1
Potar_num (4ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
1	XXXX	01	Voie13	01	Voie9
Potar_num (5ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
0	XXXX	10	Voie6	10	Voie2
Potar_num (6ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
1	XXXX	10	Voie14	10	Voie10
Potar_num (7ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
0	XXXX	11	Voie7	11	Voie3
Potar_num (8ème écriture)					
Indice	Undefined	Adresse voie	Valeur	Adresse	Valeur
1	XXXX	11	Voie15	11	Voie11

8.6 Switch : (write only 32)

Switch (32 bits)					
Switch 220pF voie 15 (1 bit)	…	Switch 220pF voie 1 (1 bit)	Switch 1nF voie 1 (1 bit)	Switch 220pF voie 0 (1 bit)	Switch 1nF voie 0 (1 bit)
31	…	3	2	1	0

Pour qu'un switch soit activé il faut que le bit soit à 1. Sur chacun des ASIC, on a le choix entre deux switchs pour chaque voie. Ce qui fait au total un nombre de 32 switchs.
Ce qui induit qu'on a la possibilité entre les combinaisons 01 et 10 tous les deux bits.

8.7 Piédestaux (write only 32)

On dispose de deux boîtiers contenant chacun 8 DAC 8 bits. A chaque accès VME on récupère deux fois huits bits. Ainsi on peut alors configurer un DAC de chaque boîtier en même temps.

Piédestaux (32 bits)								
Valeur Piédestaux n° N (8 bits)			Unused			Valeur Piédestaux n° N+8 (8 bits)		
23	…	16	15	…	8	7	…	0

Base+$(20)_{hex}$	Voie 8	Voie 0
Base+$(24)_{hex}$	Voie 9	Voie 1
Base+$(28)_{hex}$	Voie 10	Voie 2
Base+$(2C)_{hex}$	Voie 11	Voie 3
Base+$(30)_{hex}$	Voie 12	Voie 4
Base+$(34)_{hex}$	Voie 13	Voie 5
Base+$(38)_{hex}$	Voie 14	Voie 6
Base+$(3C)_{hex}$	Voie 15	Voie 7

8.8 Module de configuration des seuils (write only 32)

Il y a seize seuils à sauvegarder (16 accès VME) dans une mémoire du FPGA, en sachant que chaque seuil est défini sur 12 bits.

Seuils (32 bits)					
Undefined (20 bits)			Valeur Seuils n°N (12 bits)		
31	…	12	11	…	0

8.9 Programmation de l'interruption suite au START (write only 32)

Une écriture dans ce registre permet de sélectionner le niveau de l'interruption, et de revalider l'IT après un cycle d'interruption. Le simple fat d'accéder ce registre valide (ou revalide) l'interruption.

Interruption (32 bits)								
Undefined (21 bits)			Status_event (8 bits)			Niveau d'IT Porte (3 bits)		
31	…	11	10	…	3	2	…	0

8.10 Récapitulatif

Adresse	Registre	Mode D'accès
Base+$(00)_{hex}$	ID_CODE	Read 32
Base+$(04)_{hex}$	FIFO_Readout	Read 32
Base+$(08)_{hex}$	Data_reg_sl_sc_and_run	Write 32
Base+$(0C)_{hex}$	Interruption	Write 32
Base+$(10)_{hex}$	Non utilisé	
Base+$(14)_{hex}$	Non utilisé	
Base+$(18)_{hex}$	Potar_num	Write 32
Base+$(1C)_{hex}$	Switch	Write 32
Base+$(20)_{hex}$	Piédestaux voies 0 et 8	Write 32
Base+$(24)_{hex}$	Piédestaux voies 1 et 9	Write 32
Base+$(28)_{hex}$	Piédestaux voies 2 et 10	Write 32
Base+$(2C)_{hex}$	Piédestaux voies 3 et 11	Write 32
Base+$(30)_{hex}$	Piédestaux voies 4 et 12	Write 32
Base+$(34)_{hex}$	Piédestaux voies 5 et 13	Write 32
Base+$(38)_{hex}$	Piédestaux voies 6 et 14	Write 32
Base+$(3C)_{hex}$	Piédestaux voies 7 et 15	Write 32
Base+$(40)_{hex}$	Seuils voie 0	Write 32
Base+$(44)_{hex}$	Seuils voie 1	Write 32
Base+$(48)_{hex}$	Seuils voie 2	Write 32
Base+$(4C)_{hex}$	Seuils voie 3	Write 32
Base+$(50)_{hex}$	Seuils voie 4	Write 32
Base+$(54)_{hex}$	Seuils voie 5	Write 32
Base+$(58)_{hex}$	Seuils voie 6	Write 32
Base+$(5C)_{hex}$	Seuils voie 7	Write 32
Base+$(60)_{hex}$	Seuils voie 8	Write 32
Base+$(64)_{hex}$	Seuils voie 9	Write 32
Base+$(68)_{hex}$	Seuils voie 10	Write 32
Base+$(6C)_{hex}$	Seuils voie 11	Write 32
Base+$(70)_{hex}$	Seuils voie 12	Write 32
Base+$(74)_{hex}$	Seuils voie 13	Write 32
Base+$(78)_{hex}$	Seuils voie 14	Write 32
Base+$(7C)_{hex}$	Seuils voie 15	Write 32

8.11 Vue Générale de la CARTE QDC16

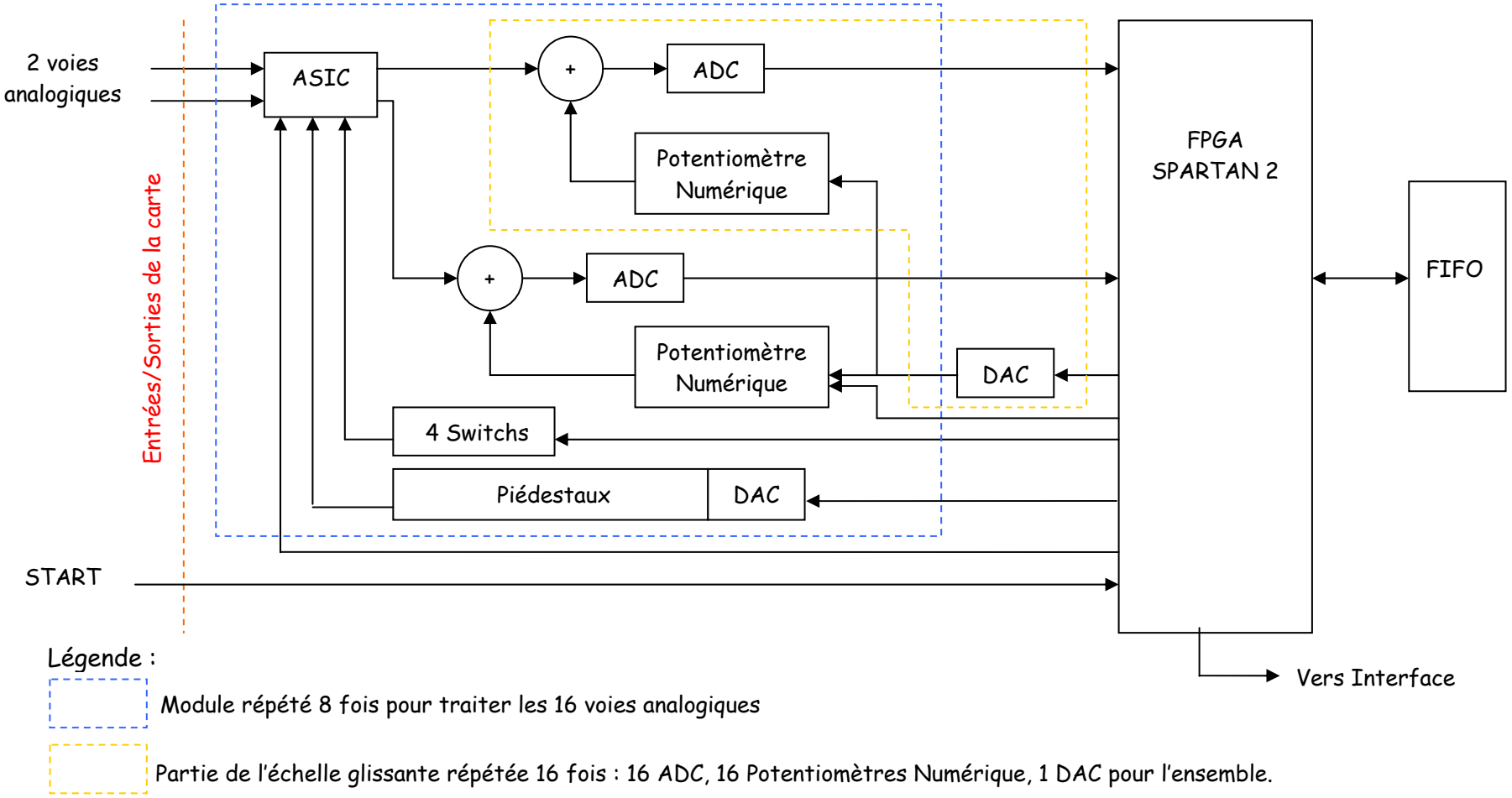


Figure 21 : Schéma de l'ensemble de la carte

8.12 Vue Générale de l'intérieur du FPGA

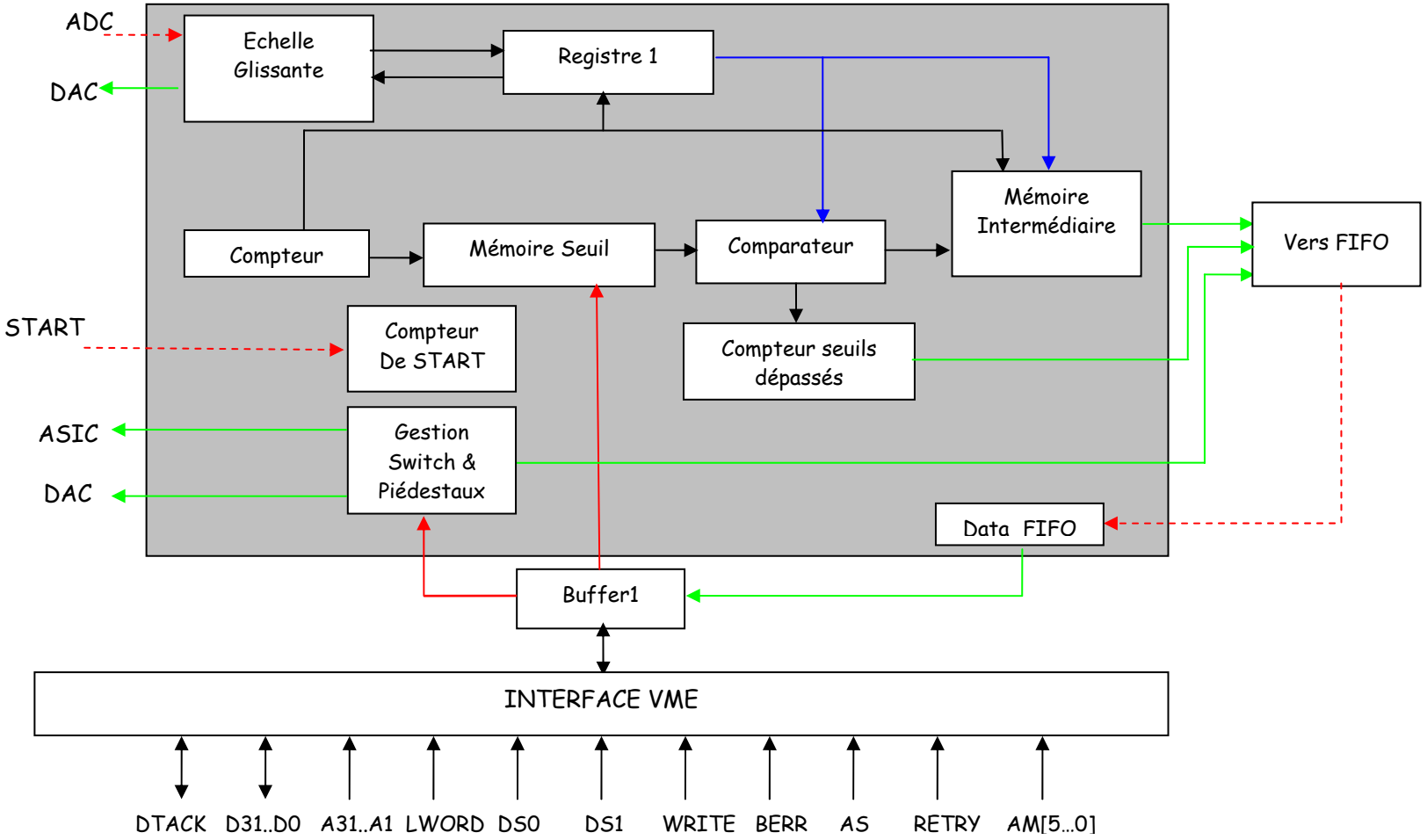


Figure 22 : Schéma descriptif des blocs à l'intérieur du FPGA

Le FPGA utilisé est le Spartan 2. Il est composé de 137 entrées sorties et est alimenté en 3,3V. Sa fréquence de travail est de 40Mhz.

8.13 Module : Échelle Glissante

Ce qui résulte de l'ADC est directement dirigé vers le FPGA dans lequel on soustraira la valeur ajoutée précédemment. Ainsi on a en notre possession une valeur numérique qui correspond bien à ce qu'on avait en entrée (mais convertit selon nos besoins).

L'échelle glissante peut être ou ne pas être activer. C'est au physicien de déterminer et de le dire à la carte. Une lecture VME nous permettra de connaître son choix.

8.13.1 L'ADC

L'ADC choisit est l'AD7476 12 bits disposant d'une sortie série. De plus il fonctionne à une fréquence maximum de 20 MHz.
La première remarque est qu'un ADC 12 bits a la capacité de convertir des valeurs comprises entre 0 et 4095. Donc en sachant qu'au résultat de l'intégration on ajoute au maximum 255, le résultat précédent ne doit pas excéder 4095 moins 255 soit 3840.
Afin de récupérer les données, il va falloir générer une horloge de 20 MHz à partir du 40 MHz du Spartan 2.
Comme on le voit sur la figure suivante, il nous faudra 16 coups d'horloge à 20 MHz pour récupérer la totalité des données. En effet, on ne doit pas tenir compte des trois premiers bits à 0 envoyés.

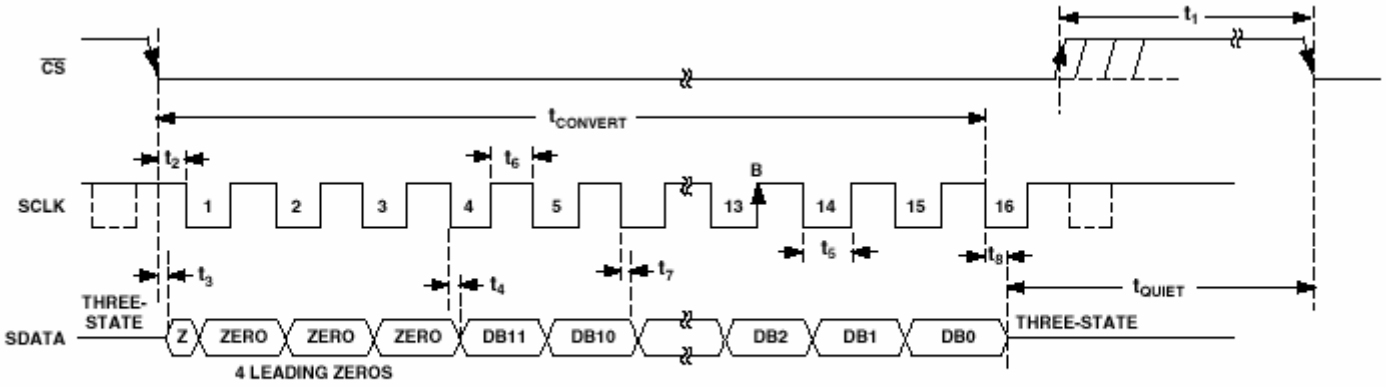


Figure 15. AD7476 Serial Interface Timing Diagram

Figure 23 : Conversion ADC

8.13.2 Le DAC

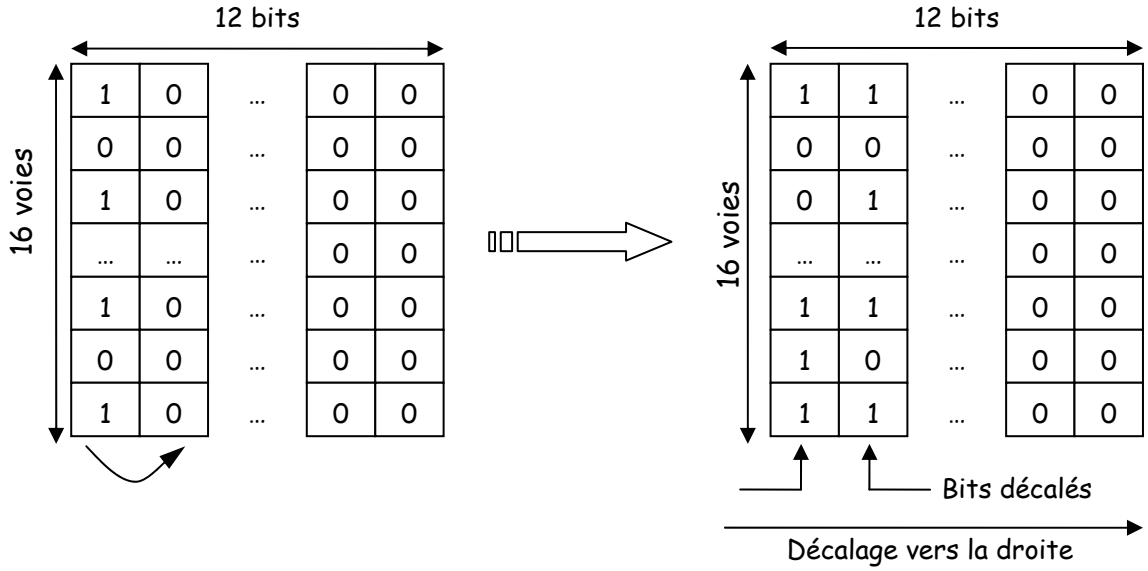
Pour ce qui est du DAC, c'est le FPGA qui contrôle la valeur à convertir. Pour cela on a créé un compteur qui envoie une valeur comprise entre 0 et 255. Ce compteur évolue à chaque START, c'est à dire à chaque nouvelle mesure. Si on est à 0, on incrémente et dès qu'on atteint 255, on décrémente.
Le DAC choisit, est l'AD7801 et qui donne le résultat de la conversion sur 8 bits. On peut donc au mieux convertir la valeur maximum de 255.

Dans le cas où l'échelle glissante n'est pas validée, le compteur n'est plus pris en compte. On réalise une lecture VME à l'adresse du registre appelé val_sl_sc_off&run (voir tableau d'adressage).

8.13.3 Le registre à décalage

A chaque coup d'horloge, on récupère un bit issu de la conversion sur chaque voie. Par conséquent, comme nous avons 16 voies, on reçoit 16 bits qu'il faut ranger.
Le but de cette machine d'état est de récupérer les valeurs issues des 16 ADC 12 bits auxquelles on a ajouté une valeur (problème de non linéarité).

Dans la registre, nous rangeons les données de la manière suivante :



Sur le schéma précédent, on aura sauvegardé les 16 valeurs sous forme binaire au bout de 15 coups d'horloge.

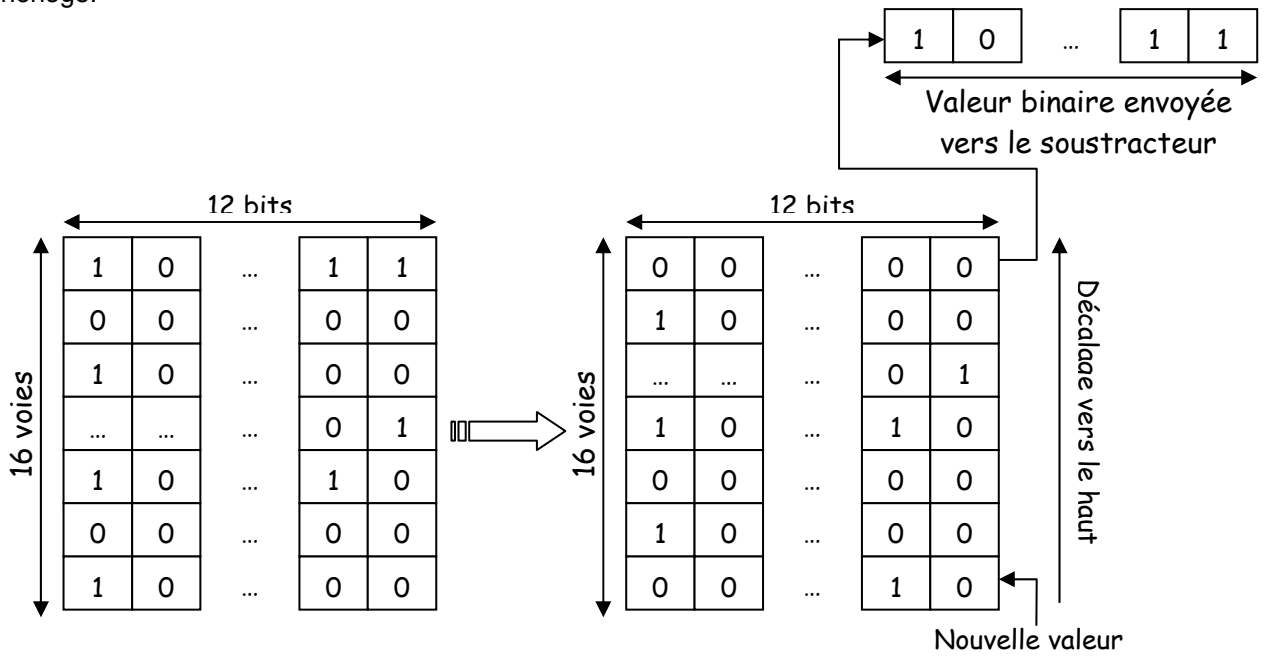


Figure 24 : Schéma décrivant le décalage dans la mémoire intermédiaire

Dans ce second schéma, on décale vers le haut car la valeur en tête de registre est envoyée vers le soustracteur.

8.14 Machine d'état : FSM d'acquisition

Pour traduire tout ce que l'on vient de dire, on se sert de machine d'état. Ces machines d'états fonctionnent comme des graphcets que l'on a étudiés.
Ces machines m'ont été demandé par mon tuteur afin de faciliter la programmation et qui s'est vérifié.

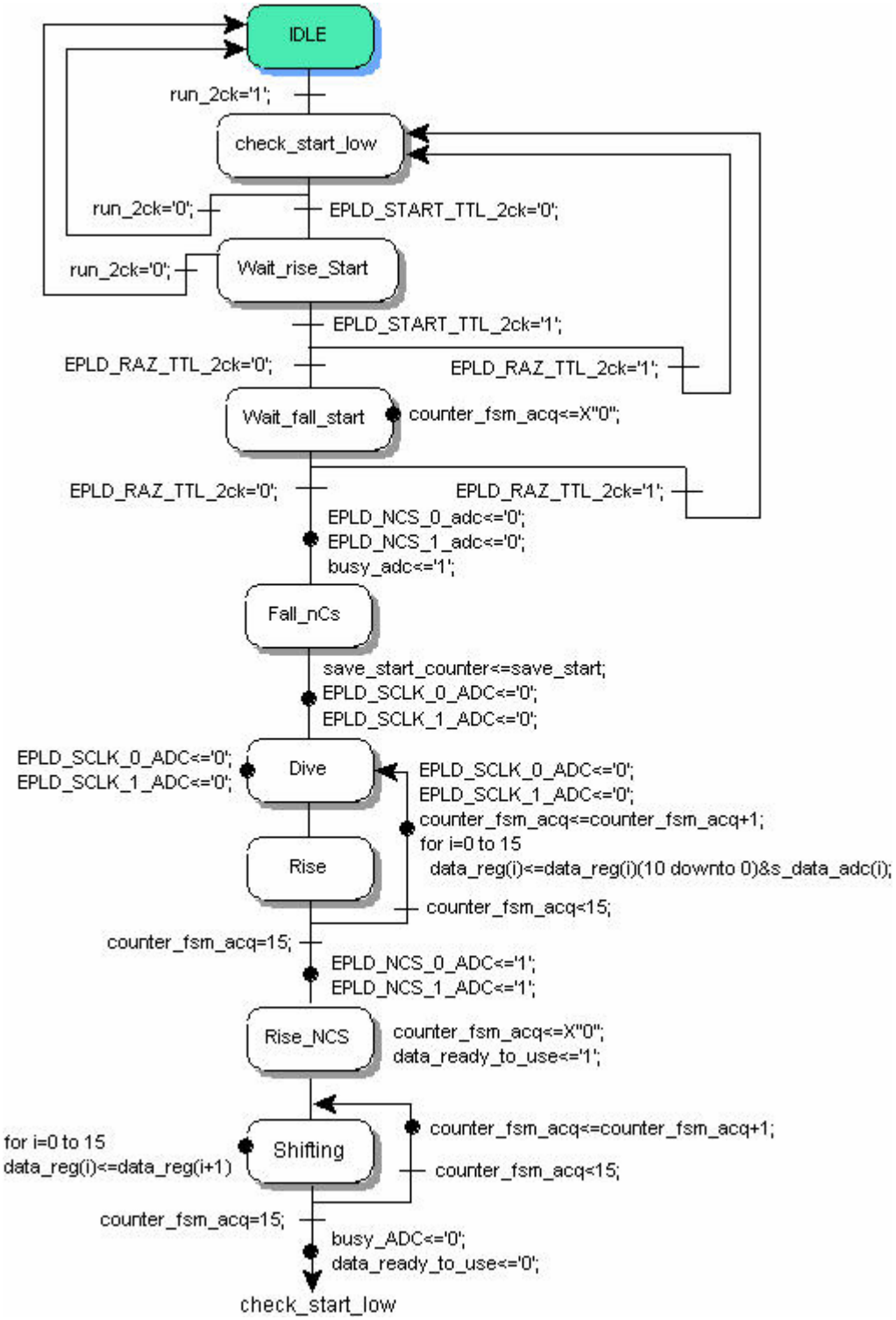


Figure 25 : FSM d'acquisition

8.15 Module de comparaison

Une fois obtenue la valeur réelle on envoie la première valeur du registre vers un comparateur. A cette valeur on compare un seuil correspondant à cette voie.
Cette comparaison permet de déterminer si la mesure est strictement supérieure au seuil. Elle est faite en trois coups d'horloge tel qu'on peut le voir à l'annexe 1. On s'aperçoit que je me suis basé sur un système de portes logiques « OU » , « ET » et de bascules. A chaque coup d'horloge, chaque bascule mémorise et au bout du troisième coup, la dernière bascule mémorise un '1' ou un '0' logique. Si c'est '1' c'est que la mesure est supérieure.

8.16 Compteur

Le compteur en sortie du comparateur sert à déterminer le nombre de données supérieures aux seuils.
Cette donnée sera envoyée dans la FIFO extérieure au Spartan 2.
L'utilité de compteur est qu'il nous permettra de connaître le nombre de lecture que l'on devra faire dans la FIFO pour récupérer et envoyer les données à la carte Maître

8.17 Synoptique du module principal

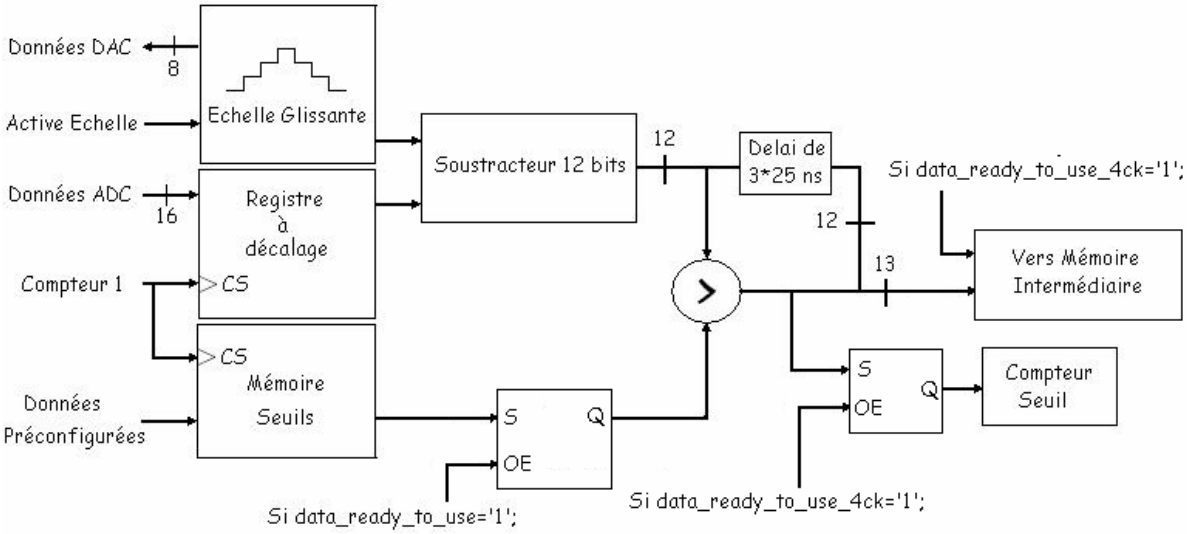


Figure 26 : synoptique du module principal

Ce synoptique illustre tous les modules afin de mieux comprendre encore le traitement successif des données.
On peut s'apercevoir d'ailleurs que le compteur du DAC est codée sur 8 bits et que l'on envoie cette valeur au soustracteur 12 bits. Pour palier à ce problème, on rajoute simplement quatre bits de poids fort à 0 ce qui ne change rien à la valeur initiale.
Le rôle du compteur 1 est aussi mis en évidence. Il permet de sélectionner le seuil et la valeur correspondante.
Le délai permettant de synchroniser l'envoi de la valeur et du bit qui symbolise le résultat de la comparaison.

8.18 Machine d'état : Échelle Glissante

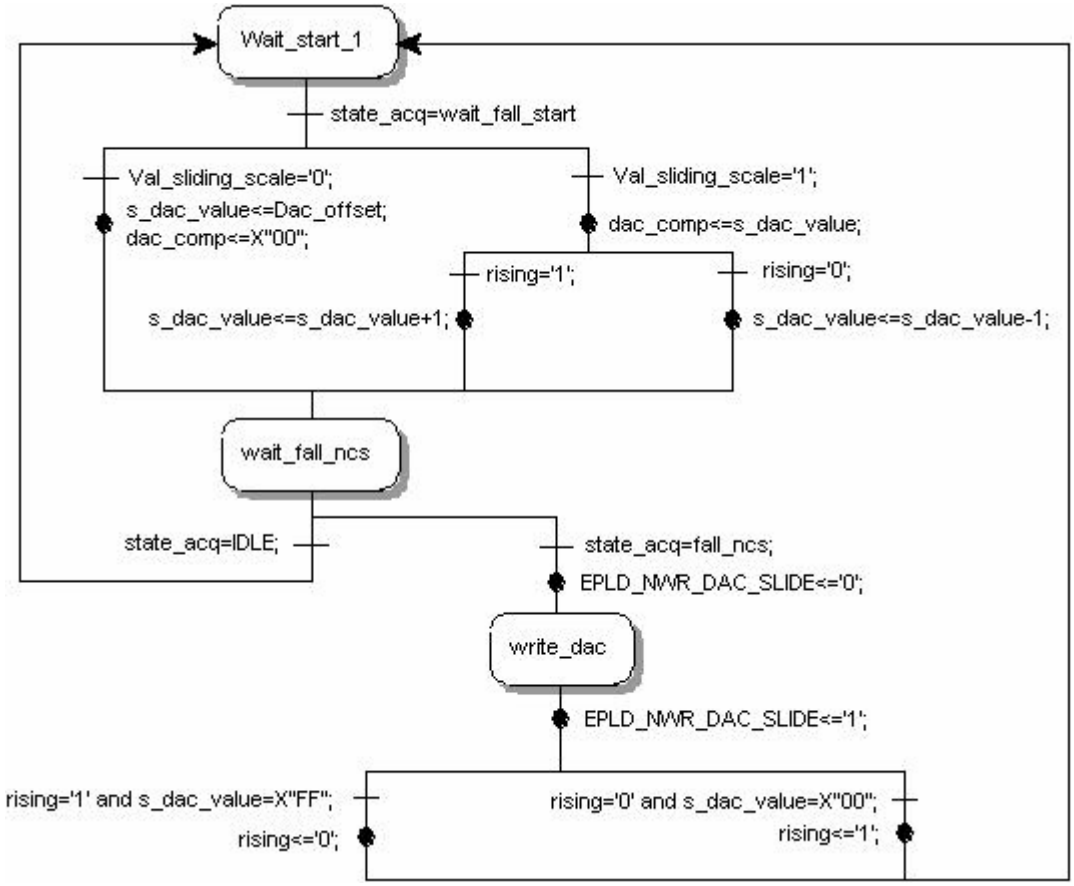


Figure 27 : FSM échelle glissante

But de la machine d'état : gestion de la valeur envoyée vers le DAC de l'échelle glissante (EPLD_DAC_SLIDE), et de celle envoyée vers le comparateur (dac_comp).
Tout d'abord, on doit déterminer si l'échelle glissante est validée ou si on est en calibration. C'est l'état de « do_cal_sl_sc » qui permet de tester et déterminer dans quelle condition on se trouve. Si elle vaut 1, soit on souhaite calibrer l'échelle glissante soit elle est désactivée. Sinon si elle vaut 0, l'échelle est activée.
Dans le cas où, l'échelle est désactivée ou en calibration on envoie zéro vers le soustracteur et on affecte au DAC la valeur contenue dans le registre « data_reg_sl_sc_and_run ».
Lors de la calibration, sachant la valeur dans ce registre, on peut visualiser si le DAC nous sort la valeur convertie correspondante. Sinon si on est en échelle désactivée et que la valeur dans le registre est différente de 0, la valeur convertie aura un offset qu'on ne retirera pas au niveau du soustracteur.
Dans le cas où l'échelle est activée, on envoie une valeur vers le DAC par l'intermédiaire d'un compteur. Cette valeur est comprise entre 0 et FF (égale à 255). A l'état initial le compteur est mis à 0 et à chaque START on incrémentera le compteur de un. Dès que l'on atteint 255 on met la variable rising à 0 ce qui nous permet de dire au système que maintenant il faut décrémenter. Et ainsi de suite, dès qu'on arrive à 0 on incrémente. A chaque fois que l'on a incrémenté ou décrémenté, on attend un START qui validera la transition après l'étape « Wait_START_1 ».

8.19 Soustraction

But : soustraire à la valeur convertie, l'offset rajouté à l'aide du DAC. Si l'échelle glissante est activée, on retire la valeur ajoutée sinon rien. Le signal dac_comp sera initialisé à 0.
La soustraction fonctionne en permanence.

8.20 Machine d'état : mémoire intermédiaire

Lorsque la comparaison est terminée, on sauvegarde le bit généré par le comparateur et la valeur dans une mémoire. La comparaison mettant trois coups d'horloge, je retarde de trois coups d'horloge l'envoi de la donnée vers cette mémoire intermédiaire

Bitmap des mots dans la mémoire intermédiaire :

12	11	…	0
Bit_Seuil_OK	Données encodées sur 12 bits		

Tableau 1 : Format des données dans la mémoire intermédiaire

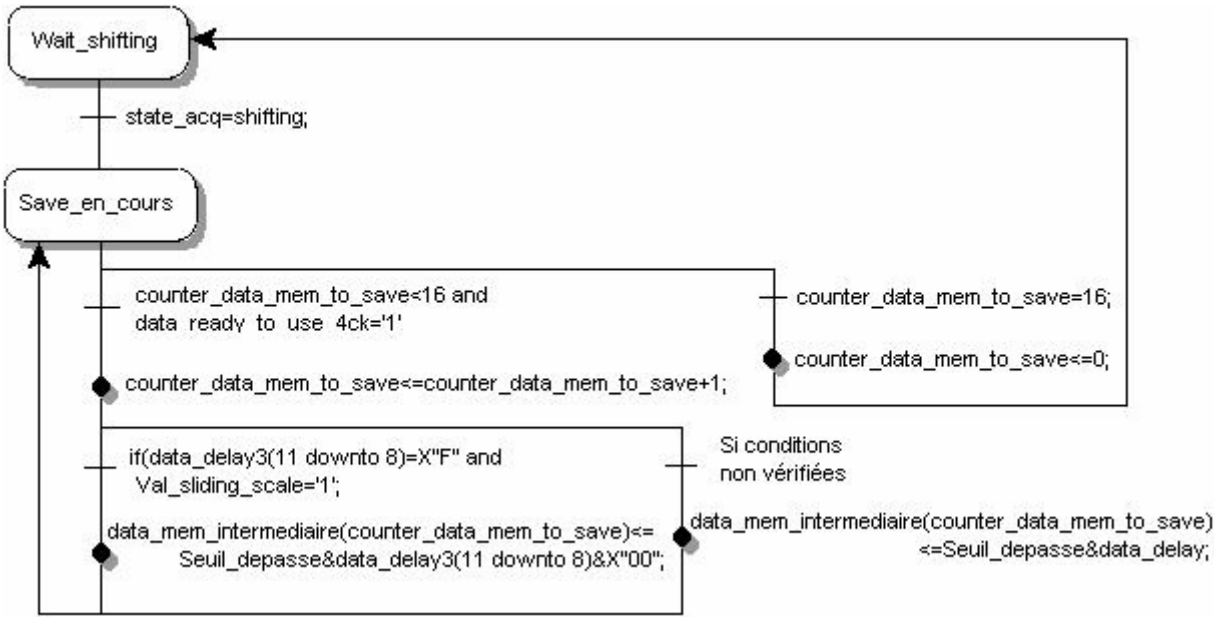
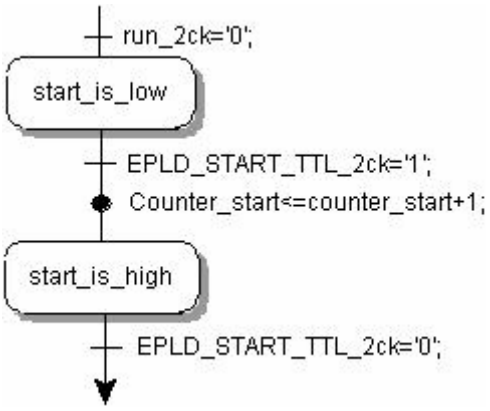


Figure 28 : FSM sauvegarde dans la mémoire intermédiaire

8.21 Machine d'état : compteur de START



On détecte le passage de 0 à 1 pour incrémenter le compteur de START.

Figure 29 : FSM compteur de START

8.22 Machine d'état : FSM d'écriture dans la FIFO

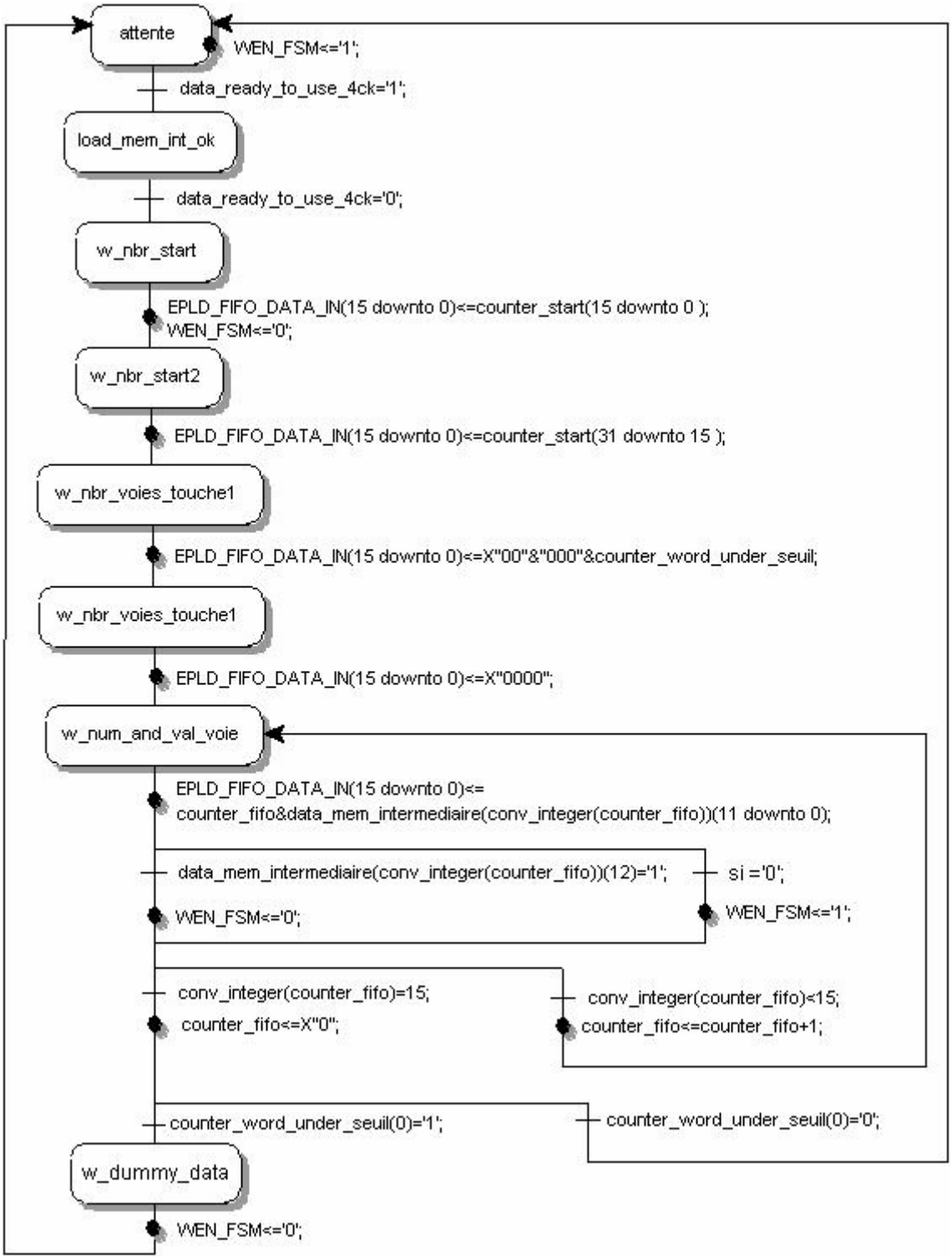


Figure 30 : FSM d'écriture dans la FIFO

8.23 Traitement FULL FIFO

Lorsque la Fifo est pleine, il faut arrêter d'envoyer des données et surtout il faut ne pas en perdre. En effet, la FIFO peut être pleine alors qu'on est en train d'envoyer des valeurs et n'avoir pour un évènement qu'un certain nombre de valeur de voie.

Afin d'obtenir ce résultat il faut avant tout bloquer la machine d'état de l'écriture de la FIFO. Ceci est fait à l'aide de l'EPLD_FIFO_FF.

Le plus dur est de gérer le Write Enable de la FIFO parce que le temps qu'on s'aperçoive que la FIFO est pleine, il nous faut un coup d'horloge pour pouvoir réagir. C'est pour ça qu'on utilise une porte « OU » :

EPLD_FIFO_WEN<= WEN_FSM or not(EPLD_FIFO_FF) ;

L'EPLD_FIFO_WEN sera forcément à l'état bas (donc écriture), quand le WEN_FSM sera à 0 et que la FIFO sera non pleine.

Dans la machine d'état on modifie l'état du WEN_FSM qui avec l'EPLD_FIFO_FF n'étant plus dépendant de l'horloge permettent à l'EPLD_FIFO_WEN de changer d'état plus rapidement.

8.24 Machine d'état : FSM de lecture de la FIFO

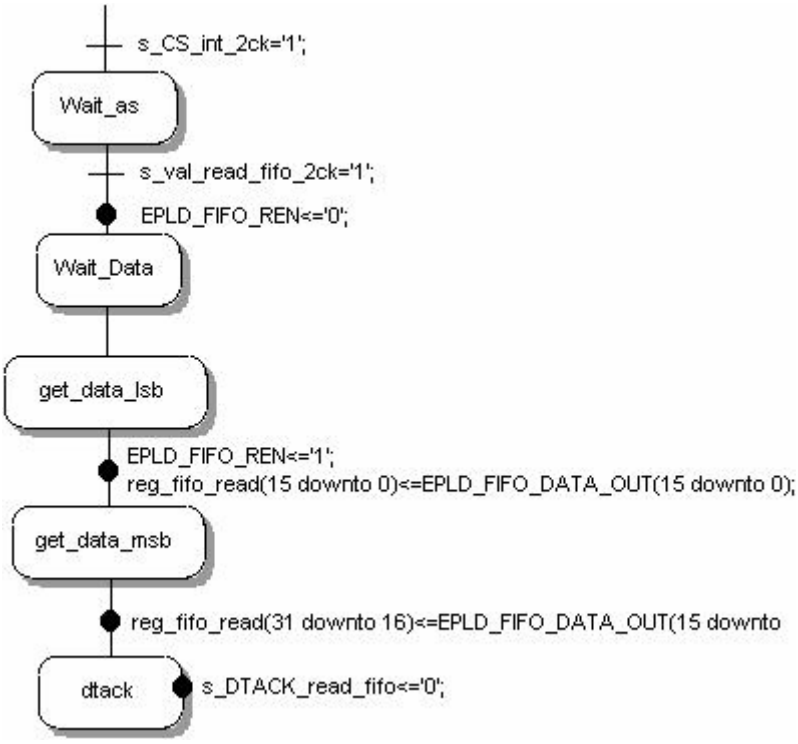


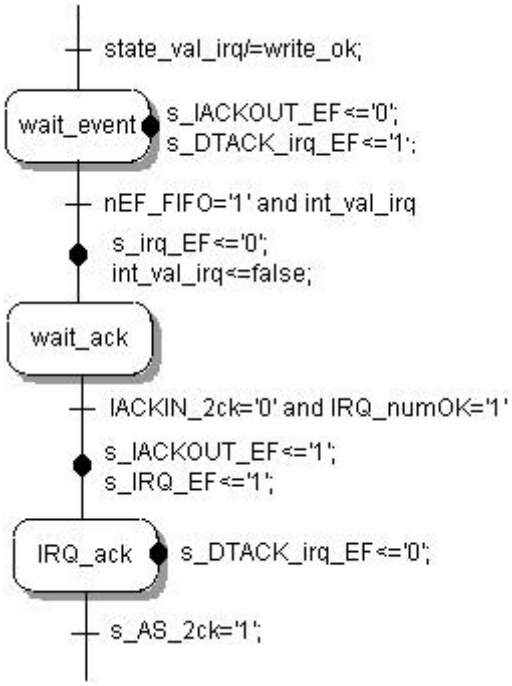
Figure 31 : FSM de la lecture dans la FIFO

8.25 Temps mort

On active le temps mort quand

- on détecte un front montant du START.
- les ADC sont en cours de conversion et ou on a un raz CCT actif.
- l'écriture dans la FIFO est en cours.

8.26 Machine d'état de Gestion des IT



s_val_irq_EF<='1' when (EPLD_VME_ADD(6 downto 1)= « 000110 » and EPLD_VME_WRITE='0') ; else '0' ;

Figure 32 : FSM de gestion des IT

9 Architecture de QDC16_VME

Les adresses sont données de A31 à A1. A0 est supposée constamment nul, et donc il faut multiplier par 2 les adresses suivantes.

9.1 Interface du FPGA

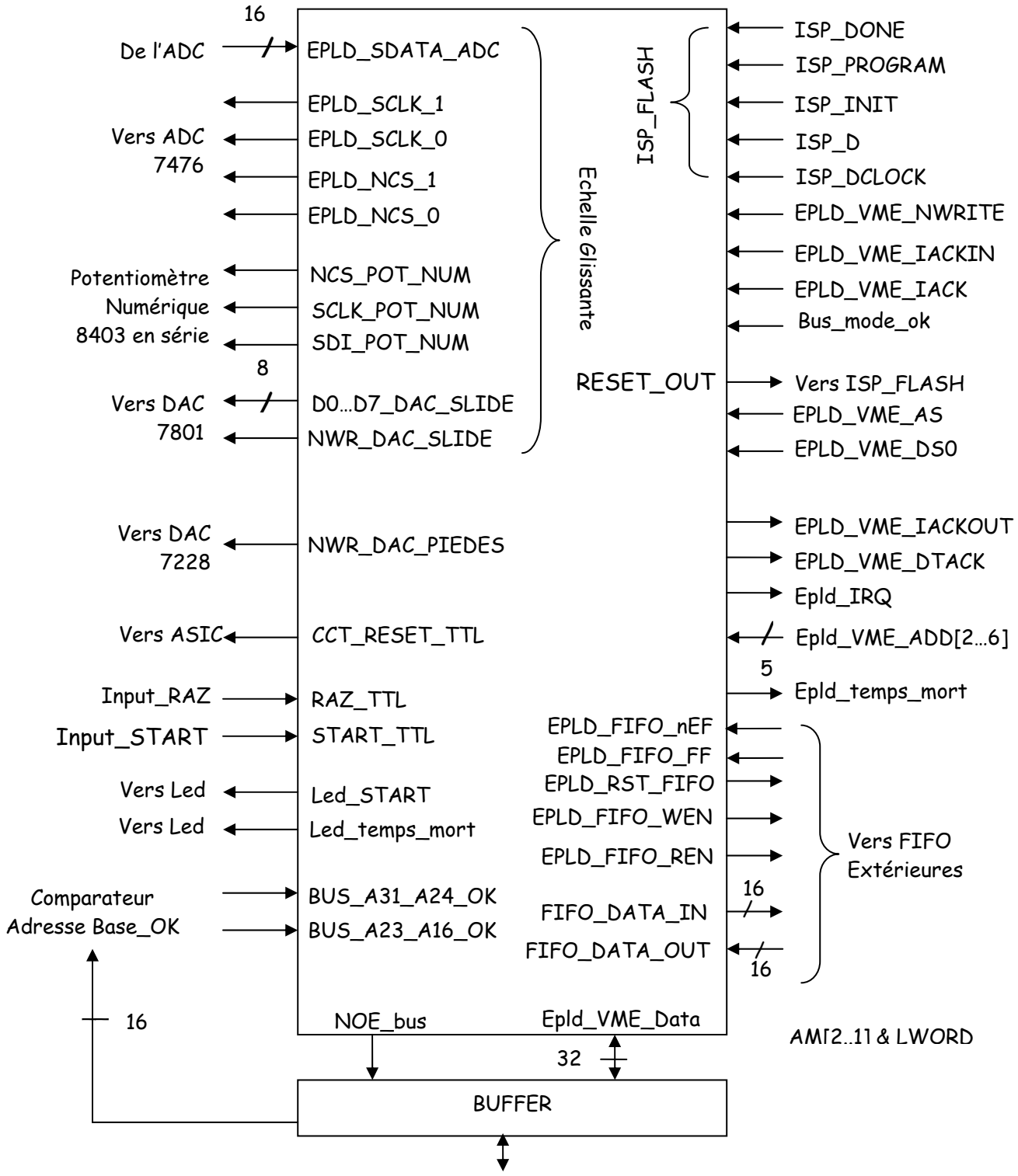


Figure 33 : Schéma d'entrées / sorties du FPGA

9.2 FSM IDCODE

Dès que l'adresse est vérifiée et correcte, on met le DTACK à 0. L'esclave indique ainsi au maître qu'il a reçu avec succès les données.

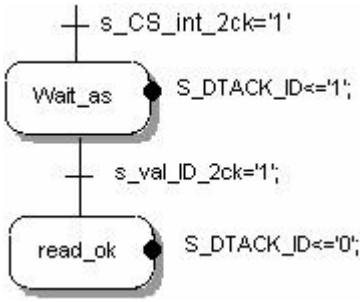


Figure 34 : FSM IDCODE

Base +0 soit $(0000\ 000)_{bin} \rightarrow (00)_{hex}$
S_val_ID<='1' when (EPLD_VME_ADD(6 downto 1)= « 000000 » and EPLD_VME_WRITE='1') else '0' ;
IDCODE_POLLING(31)<=IDCODE(31) or nEF_FIFO;
IDCODE_POLLING(30 downto 0)<=IDCODE(30 downto 0);

9.3 FIFO READOUT

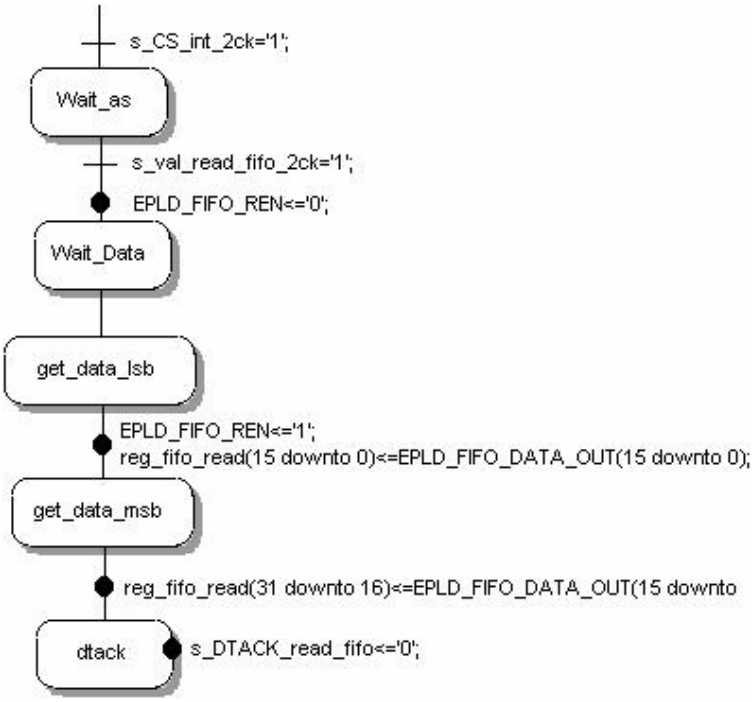


Figure 35 : FSM ReadOut

Base + 2 soit $(000\ 010)_{bin}$ (soit pour sur 32 bits base + 4) $\rightarrow (04)_{hex}$
s_val_read_fifo<='1' when (EPLD_VME_ADD (6 downto 1) = « 000010 » and EPLD_VME_WRITE='1') else '0' ;

9.4 FSM Reg_sl_sc_and_run

On n'est pas obligé d'utiliser l'échelle glissante mais on a la possibilité d'envoyer une valeur différente ou égale à 0 vers le DAC. Cette valeur peut être interprété comme un offset.

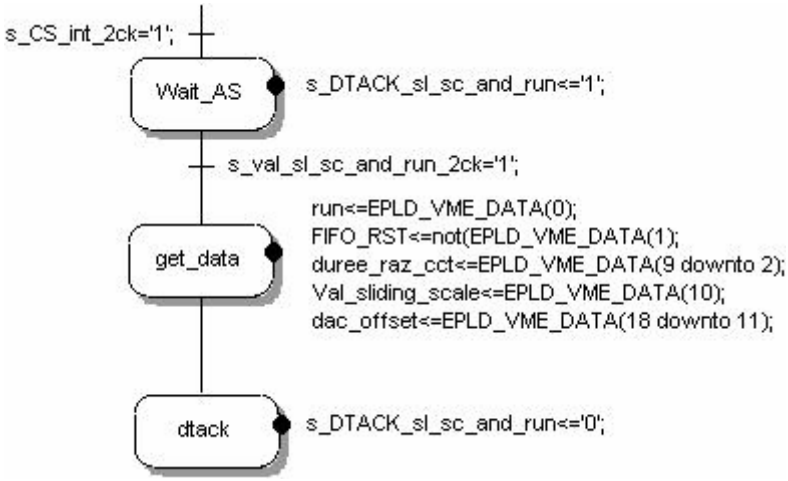


Figure 36 : FSM du registre gérant le run et l'échelle glissante

Base + 4 soit $(000\ 100)_{bin}$ (soit pour sur 32 bits base + 8) → $(08)_{hex}$
s_val_sl_sc_and_run<='1' when (EPLD_VME_ADD(6 downto 1)= « 000100 » and EPLD_VME_WRITE='0') else '0' ;

9.5 FSM Configuration Potentiomètre Numérique

16*(8+2)=160 bits au total à envoyer pour calibrer les 16 potentiomètres. Toutes les valeurs viennent du VME et à chaque écriture VME 32 bits, on reçoit deux adresses et deux valeurs correspondantes. Au bout de deux accès VME on a alors en notre possession quatre valeurs. En sachant que je ne peux configurer dans chacun des boîtiers qu'un seul potentiomètre à la fois. Ces quarantes bits sont mémorisés dans un registre 40 bits du FPGA. L'indice envoyé permet de savoir si je dois placer les données reçues dans la partie des LSB (indice : bit à 0) ou des MSB (indice : bit à 1) du registre.
Un point important est le fait que chaque boîtier est lié au suivant. C'est-à-dire que le premier bit envoyé, traverse chaque boîtier en se décalant dans les latch et servira donc à configurer le potentiomètre du dernier boîtier.

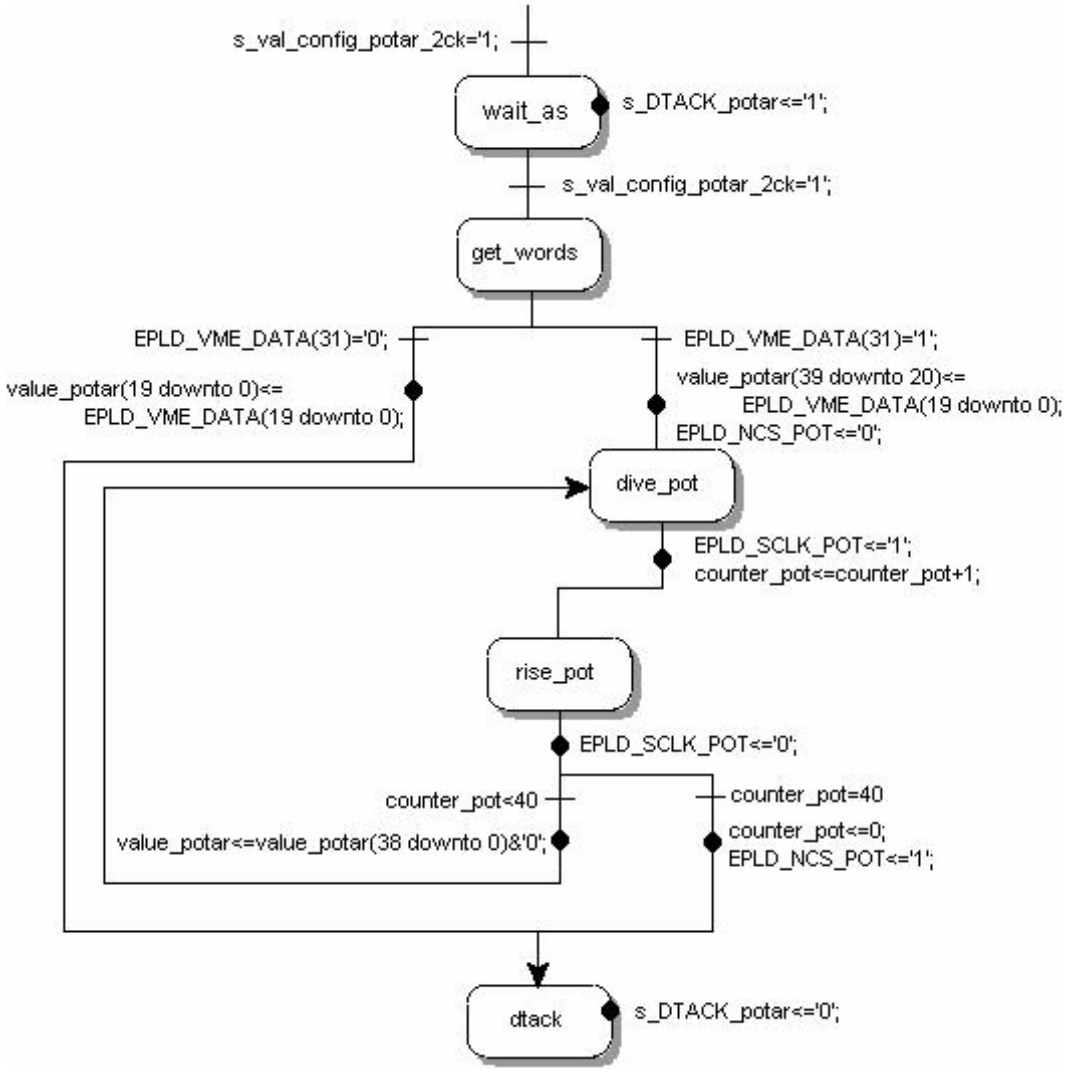


Figure 37 : FSM de configuration des potentiomètres numériques

Base + 12 soit $(001\ 100)_{bin}$ (soit pour sur 32 bits base + 24) → $(18)_{hex}$
s_val_config_potar<='1' when (EPLD_VME_ADD(6 downto 1)= « 001100 » and EPLD_VME_WRITE='0') else '0' ;

9.6 FSM de Configuration des Switchs

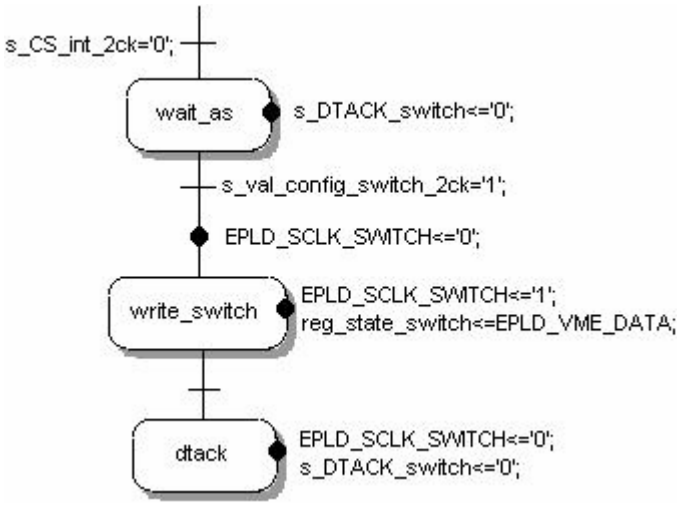


Figure 38 : FSM de configuration des switchs

Base + 14 soit $(001\ 110)_{bin}$ (soit pour sur 32 bits base + 28) → $(1C)_{hex}$
s_val_config_switch<='1' when (EPLD_VME_ADD(6 downto 1)= « 001110 » and EPLD_VME_WRITE='0') else '0' ;

9.7 FSM de Configuration des Piédestaux

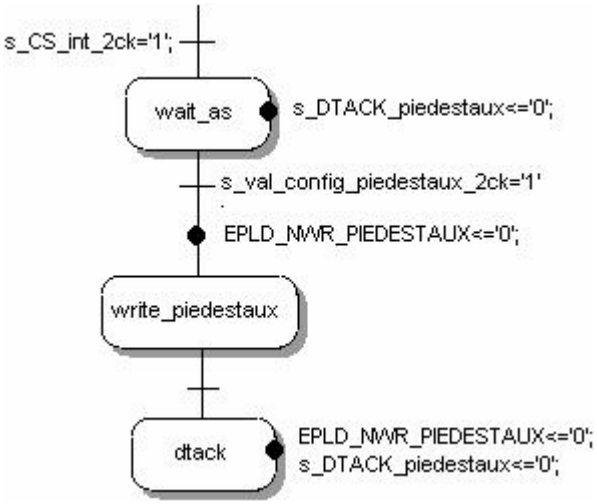


Figure 39 : FSM de configuration des piédestaux

Base + 16 soit $(010\ 000)_{bin}$ (soit pour sur 32 bits base + 32) → $(20)_{hex}$
Base + 18 soit $(010\ 010)_{bin}$ (soit pour sur 32 bits base + 36) → $(24)_{hex}$
Base + 20 soit $(010\ 100)_{bin}$ (soit pour sur 32 bits base + 40) → $(28)_{hex}$
Base + 22 soit $(010\ 110)_{bin}$ (soit pour sur 32 bits base + 44) → $(2C)_{hex}$
Base + 24 soit $(011\ 000)_{bin}$ (soit pour sur 32 bits base + 48) → $(30)_{hex}$
Base + 26 soit $(011\ 010)_{bin}$ (soit pour sur 32 bits base + 52) → $(34)_{hex}$
Base + 28 soit $(011\ 100)_{bin}$ (soit pour sur 32 bits base + 56) → $(38)_{hex}$
Base + 30 soit $(011\ 110)_{bin}$ (soit pour sur 32 bits base + 60) → $(3C)_{hex}$

s_val_config_piedestaux<='1' when (EPLD_VME_ADD(6 downto 5)= « 01 » and EPLD_VME_ADD(1)= « 0 » and EPLD_VME_WRITE='0') else '0' ;

9.8 FSM de Configuration des Seuils

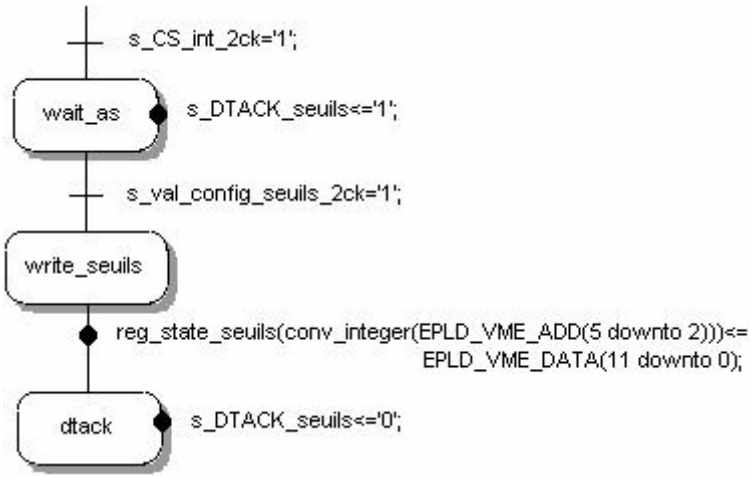


Figure 40 : FSM de configuration des seuils

Base + 32 soit (100 000)$_{bin}$ (soit pour sur 32 bits base + 64) → (40)$_{hex}$
Base + 34 soit (100 010)$_{bin}$ (soit pour sur 32 bits base + 64) → (44)$_{hex}$
Base + 36 soit (100 100)$_{bin}$ (soit pour sur 32 bits base + 64) → (48)$_{hex}$
Base + 38 soit (100 110)$_{bin}$ (soit pour sur 32 bits base + 64) → (4C)$_{hex}$
Base + 40 soit (101 000)$_{bin}$ (soit pour sur 32 bits base + 64) → (50)$_{hex}$
Base + 42 soit (101 010)$_{bin}$ (soit pour sur 32 bits base + 64) → (54)$_{hex}$
Base + 44 soit (101 100)$_{bin}$ (soit pour sur 32 bits base + 64) → (58)$_{hex}$
Base + 46 soit (101 110)$_{bin}$ (soit pour sur 32 bits base + 64) → (5C)$_{hex}$
Base + 48 soit (110 000)$_{bin}$ (soit pour sur 32 bits base + 64) → (60)$_{hex}$
Base + 50 soit (110 010)$_{bin}$ (soit pour sur 32 bits base + 64) → (64)$_{hex}$
Base + 52 soit (110 100)$_{bin}$ (soit pour sur 32 bits base + 64) → (68)$_{hex}$
Base + 54 soit (110 110)$_{bin}$ (soit pour sur 32 bits base + 64) → (6C)$_{hex}$
Base + 56 soit (111 000)$_{bin}$ (soit pour sur 32 bits base + 64) → (70)$_{hex}$
Base + 58 soit (111 010)$_{bin}$ (soit pour sur 32 bits base + 64) → (74)$_{hex}$
Base + 60 soit (111 100)$_{bin}$ (soit pour sur 32 bits base + 64) → (78)$_{hex}$
Base + 62 soit (111 110)$_{bin}$ (soit pour sur 32 bits base + 64) → (7C)$_{hex}$

s_val_config_seuils<='1' when (EPLD_VME_ADD(6)='1' and EPLD_VME_WRITE='0') else '0';

9.9 Chaînage des IT

Un retour est volontairement ajouté sur IACKIN pour avoir le temps de faire le traitement avant de propager vers IACKOUT

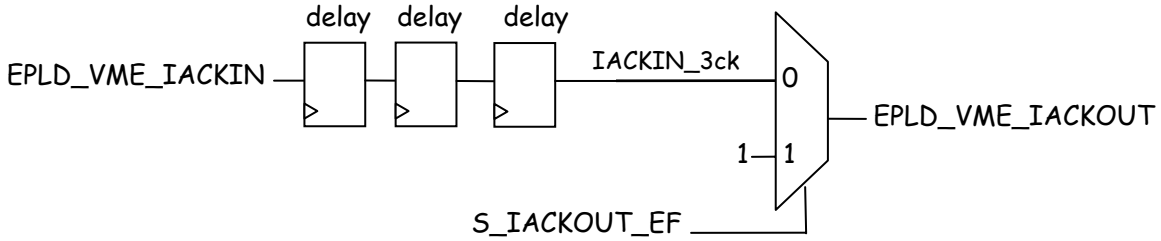


Figure 41 : gestion du chaînage des IT

EPLD_IRQ<=not(s_irq_EF) ;
IRQ_numOK<='1' when (EPLD_VME_ADD(3 downto 1)=IT_EF_level and EPLD_VME_AS='0') else '0' ;
Ainsi si la carte ne répond pas à l'interruption, elle transmet le '0', sinon elle transmet '1' et répond à l'IT.

9.10 Monostables d'activité

Pour avoir une visualisation de l'arrivée des événements et de la mise en temps mort de la carte, il faut allumer des LED pendant un minimum de 100 ms pour pouvoir les voir … La détection d'évènement se fera sur front et un compteur redéclenchable est activé :

Le compteur doit compter sur 17 bits. 2^{17}*1µs=131ms

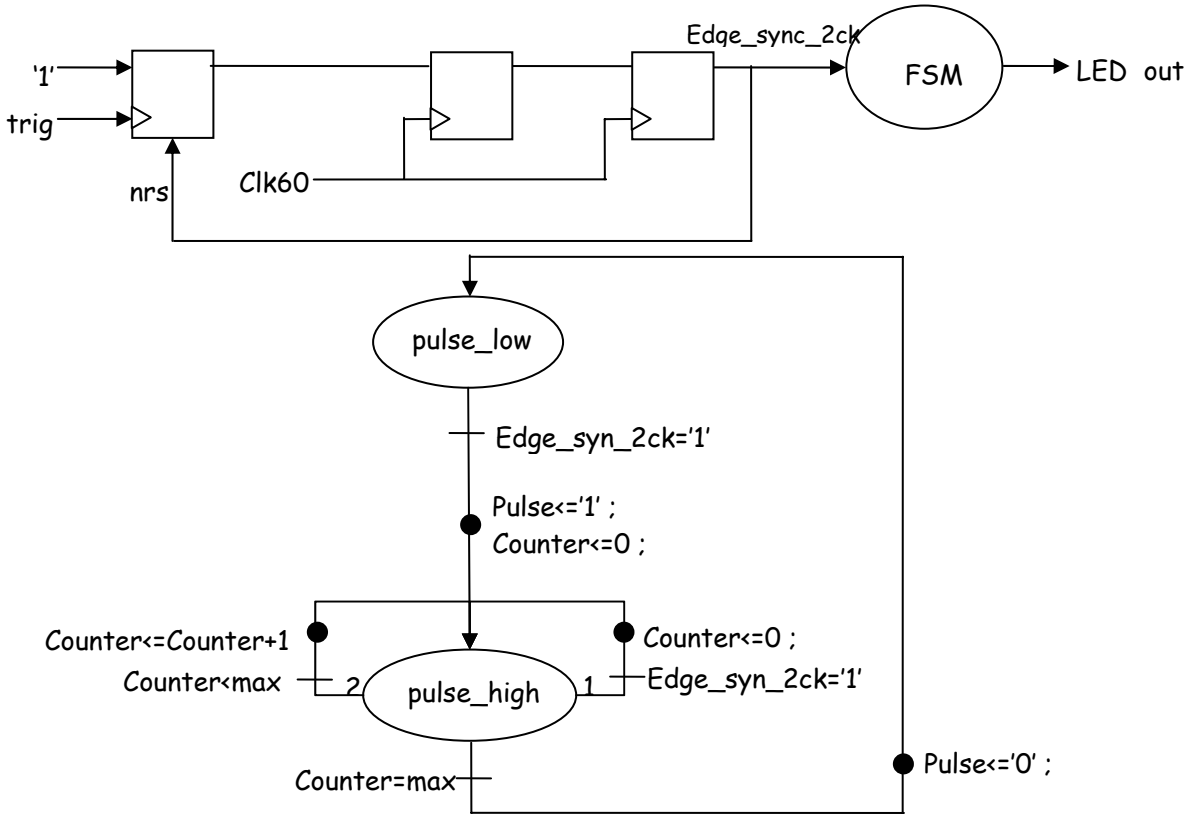
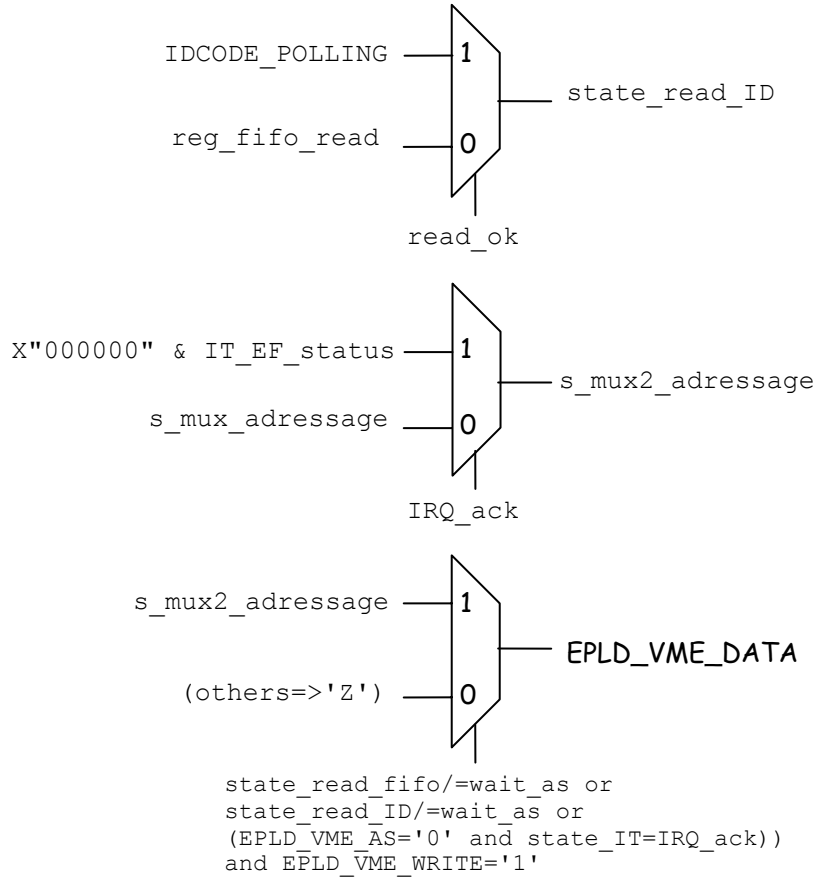
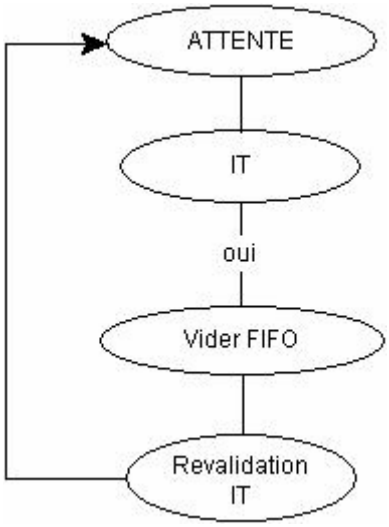


Figure 42 : description fonctionnement monostables

9.11 Multiplexage



9.12 Interruption



Une interruption est générée quand la fifo est non pleine. Dès que celle-ci est générée, on attend que l'utilisateur vide la fifo pour permettre à nouveau d'y écrire.

10 Bibliographie

Ce rapport est une compilation du :

Rapport de Mr KOUBAA Mehdi (stagiaire Master 1 EEATS) sur la partie Hardware de la carte QDC16 (Avril 2004 →juin 2004).

Et du rapport de Mr CERRI Jean-Christophe (stagiaire Master 1 EEATS) sur la partie software de la carte QDC16 (Avril 2004 →septembre 2004).

Cette compilation a été supervisée par Mr BOURRION O.

Documents Techniques :
1) Article publié dans : **1996 IEEE Nuclear Science Symposium vol 1**
Del Guerra A.

A BiCMOS integrated charge to amplitude converter
Année d'édition : 1997
Auteurs : Gallin-Martel L., Pouxe J., Rossette O.

Thèmes : Instrumentation et technologie
Sous-thèmes : Autres techniques
Editeur : Institute of Electrical and Electronics Engineers

Langue: Anglais
ISN : 0780335341
Sigle : ISN 96167
Pagination : 412-416

2) Congrès
1996 IEEE Nuclear Science Symposium
Lieu : Anaheim California Etats-Unis Dates : 02/11/1996 09/11/1996

Laboratoire de Physique Subatomique et de Cosmologie
Auteurs du Laboratoire : **Gallin-Martel L., Pouxe J., Rossetto O.**

3) Documentation ISPFLASH réalisée par Mr BOURRION O : Rapport interne ***LPSC 0323***, septembre 2003.